GOING SERVERLESS: AN INTRODUCTION TO AWS LAMBDA

BY ALEX WILSON

A LITTLE ABOUT ME

Name: Alex Wilson

Role: Engineering at T. Rowe Price

Technology stack:

- Front end:
  - Angular JavaScript Framework
- Backend:
  - AWS Dynamo DB
  - Java Spring RESTful Services

I also like to code in React, ensure website accessibility, work on open source projects, and give talks and lectures like this one!

You can find me by going to my website: CodeByAlex.com

Or follow me on Twitter and Github: @CodeByAlex

WHAT IS SERVERLESS COMPUTING?

Serverless computing allows you to build and run applications and services without thinking about managing the servers yourself.

Your application still runs on servers, but all the server management is done by an outside entity.

Serverless computing saves you money because your server is not running at all times

Let's be real…the name is misleading

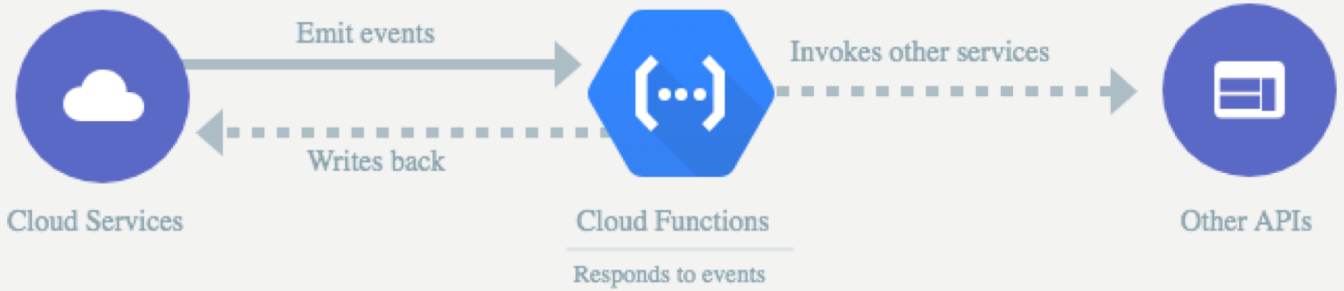
WHAT IS A SERVERLESS FUNCTION OR A FUNCTION AS A SERVICE (FAAS)?

A Serverless function is a single function that can be readily made available when you need it from anywhere in the world

The "server" suddenly appears when you need it, then disappears when you're done with it.

Now you can think in terms of functions instead of servers, and all your business logic can now live within these functions.

Allows developers to focus on their code rather than to think about the multi-threading or load-balancing of their servers. They trust the FaaS to handle all the resource management for them.



FUNCTIONS AS A SERVICE (FAAS) PLATFORMS

- AWS Lambda
- Azure Functions
- Google Cloud Functions
- IBM OpenWhisk
- Fission.io

fission

WHAT IS AWS?

Amazon Web Services (AWS) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. – Amazon

It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.

AWS LAMBDA

The code you run on AWS Lambda is uploaded as a "Lambda function".

Each function has associated configuration information, such as its name, description, entry point, and resource requirements.

Lambda functions can include libraries, even native ones.

You pay only for the compute time that you consume — there is no charge when your code is not running.

With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

HOW DO YOU TRIGGER AN AWS LAMBDA FUNCTION?

- An HTTP request using an API Gateway (What we are going to use today)
- When creating, removing, or updating a file in Amazon Simple Storage Service (S3), an event can be triggered to call a specified lambda function.
- A notification from AWS Simple Notification Service (SNS).

TO HAVE STATE…OR NOT TO HAVE STATE

For a function to have state means that the instance of that function has properties that you can refer back to and the values will remain attached to those properties there.

Keeping functions stateless enables AWS Lambda to rapidly launch as many copies of the function as needed to scale to the rate of incoming events.

While AWS Lambda's programming model is stateless, your code can access stateful data by calling other web services, such as Amazon S3 or Amazon DynamoDB.

WHAT CODING LANGUAGES RUN IN AN AWS LAMBDA FUNCTION?

- Java
- Node.js
- Python
- C#
- PowerShell
- Go
- Ruby
- Provides a Runtime API which allows you to use any additional programming languages to author your functions

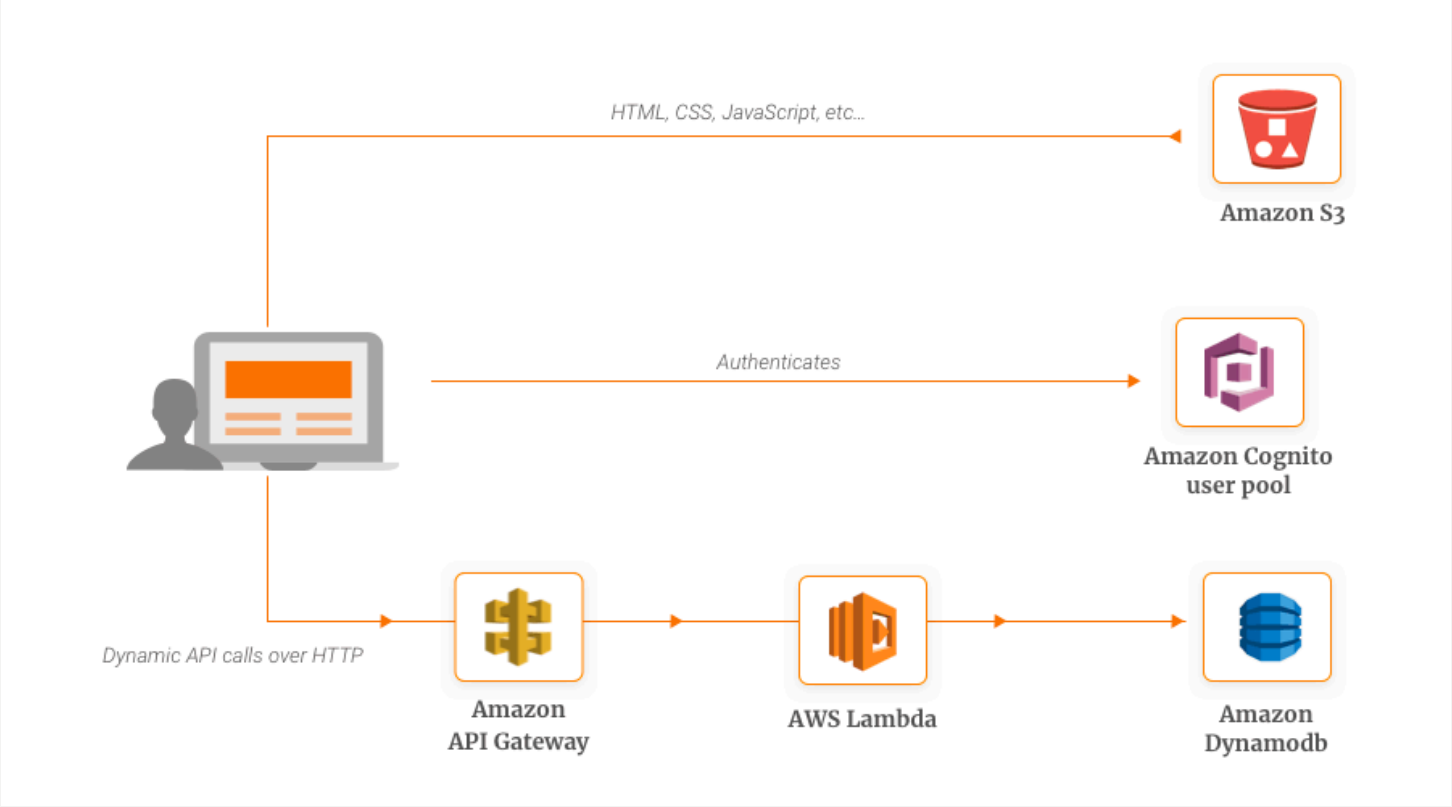
WHAT ARE THE USE CASES FOR AN AWS LAMBDA SERVERLESS FUNCTION?

- Chat bots: scaling automatically for peak demands
- Hackathon projects that need to be set up quickly with cloud based compute
- Create useful IOT functions that could be used with Google Home, Alexa, and other IOT devices
- Batch jobs scheduled tasks: Jobs that require intense parallel computation, IO or network access
- Continuous integration pipeline: The ability to remove the need for pre-provisioned hosts
- Send mass emails

In the next set of slides, we will dive deeper into some of these use cases

USE CASE #0: SERVERLESS WEBSITE EXAMPLE WITH AWS LAMBDA

- JavaScript in the browser exchanges the data from a backend API built through API Gateway and AWS Lambda.
- DynamoDB is a NoSQL database which is used for storing data through API's Lambda function.
- Amazon S3 is used for hosting the static website content like HTML, media files, CSS, JavaScript which acts as a front end in the user's browser.
- Amazon Cognito is used for user authentication and management with the help of secured backend API.

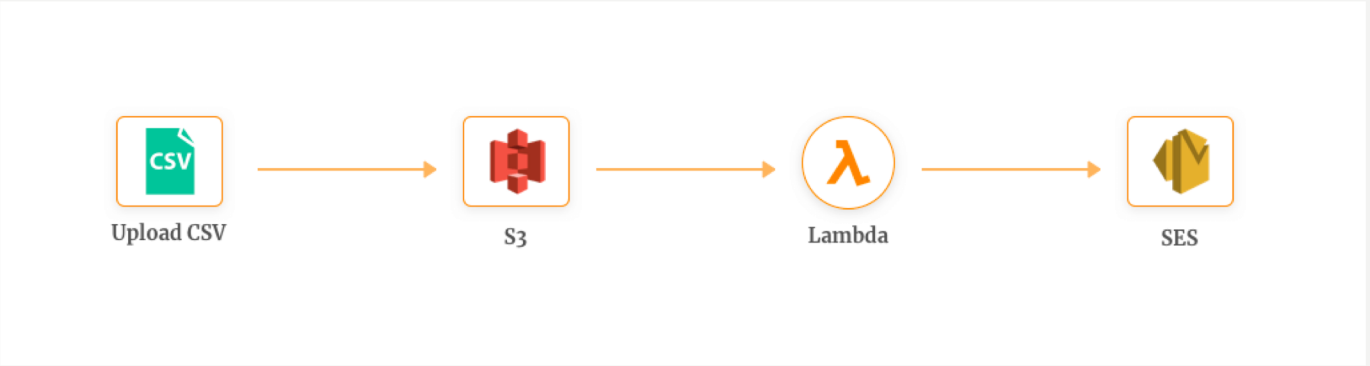


Use case from: https://www.simform.com/serverless-examples-aws-lambda-use-cases/

USE CASE #1: MASS EMAILING USING AWS LAMBDA & SES

With AWS Lambda and Simple Email Service SES, you can build a cost-effective and in-house serverless email platform. Along with S3 (where your mailing list will be stored) you can quickly send HTML or text-based emails to a large number of recipients.

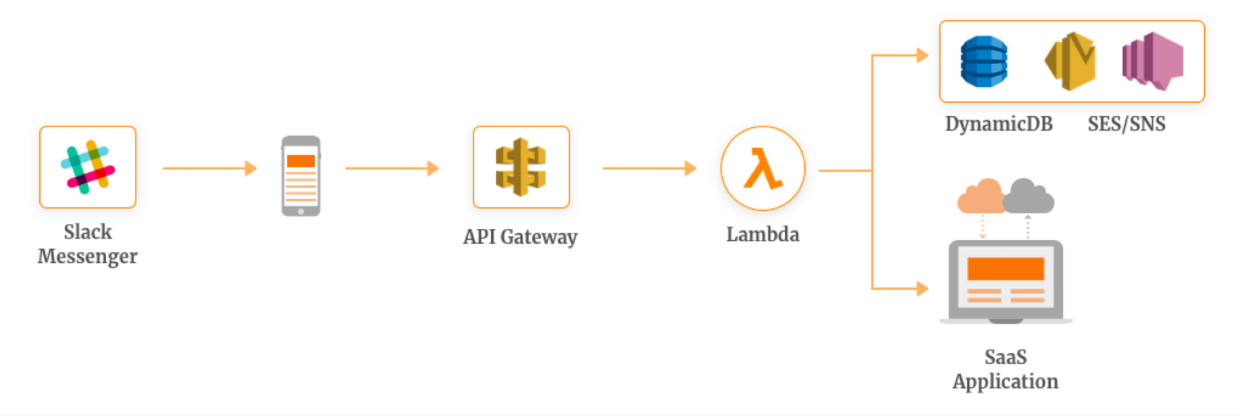
Whenever a user uploads a CSV file, it triggers an S3 event. This event triggers another Lambda function which imports the file into the database and will start sending email to all the addresses. For sending our scheduled newsletters, you can integrate it with CloudWatch Events.



Use case from: https://www.simform.com/serverless-examples-aws-lambda-use-cases/

USE CASE #2: AWS LAMBDA USE CASE FOR BUILDING SERVERLESS CHATBOT

- Building and running chatbots is not only time consuming but expensive also. Developers must provision, run and scale the infrastructural resources that run the chatbot code. However, with AWS Lambda you can run a scalable chatbot architecture.
- Enter your code logic to the Lambda function.
- Set up your code to trigger when user commands are sent to the bot. The commands are API requests (from Slack, Messenger, etc.) routed through API Gateway to Lambda function.
- Lambda runs only when it is commanded and hence using the resources when needed. You pay for the time it runs your code.



Use case from: https://www.simform.com/serverless-examples-aws-lambda-use-cases/

LET'S BUILD AN AWS LAMBDA FUNCTION!

- You will need:
  - An Amazon/AWS account
  - These slides to go at your own pace:
    - http://codebyalex.com/assets/AWS_Lambda_Lecture.pdf

- You will need to know (I will briefly go over both of these):
  - A coding language (today we will be using Node)
  - REST and RESTful API Methods
  - AWS Lambda Pricing information (we will be using the free tier)

NODE.JS: A BRIEF OVERVIEW

- A free open source server environment
- Runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Built on Chrome's JavaScript run-time for building fast and scalable network applications.
- Uses JavaScript on the server
- Asynchronous and event driven
- Single Threaded but highly scalable

* If you understand JavaScript and computer science basics, you should be well off for today's exercise

REST AND RESTFUL API METHODS

REST

- REST stands for Representational State Transfer
- Allows the transfer of data objects from one entity to another over HTTP
- Language-independent architectural style given that the language can make web-based requests using HTTP

RESTful API Methods

- GET – Used to read data from a data source
- PUT – Used to update and replace data to a data source
- POST – Used to create and save new data objects to a data source
- DELETE – Used to delete data from a data source

LAMBDA FUNCTION PRICING INFORMATION

Lambda Pricing Details

Lambda counts a **request** each time it starts executing in response to an event notification or invoke call, including test invokes from the console. You are charged for the total number of requests across all your functions.

Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 100ms. The price depends on the amount of memory you allocate to your function.

The Lambda free tier includes **1M free requests per month** and **400,000 GB-seconds of compute time per month**.

Free Tier

1M REQUESTS

per month

400,000 GB-SECONDS

of compute time per month.

The Lambda free tier does not automatically expire at the end of your 12 month AWS Free Tier term, but is available to both existing and new AWS customers indefinitely.

Requests

1M REQUESTS FREE

First 1M requests per month are free.

$0.20 PER 1M REQUESTS THEREAFTER

$0.0000002 per request.

Duration

400,000 GB-SECONDS PER MONTH FREE

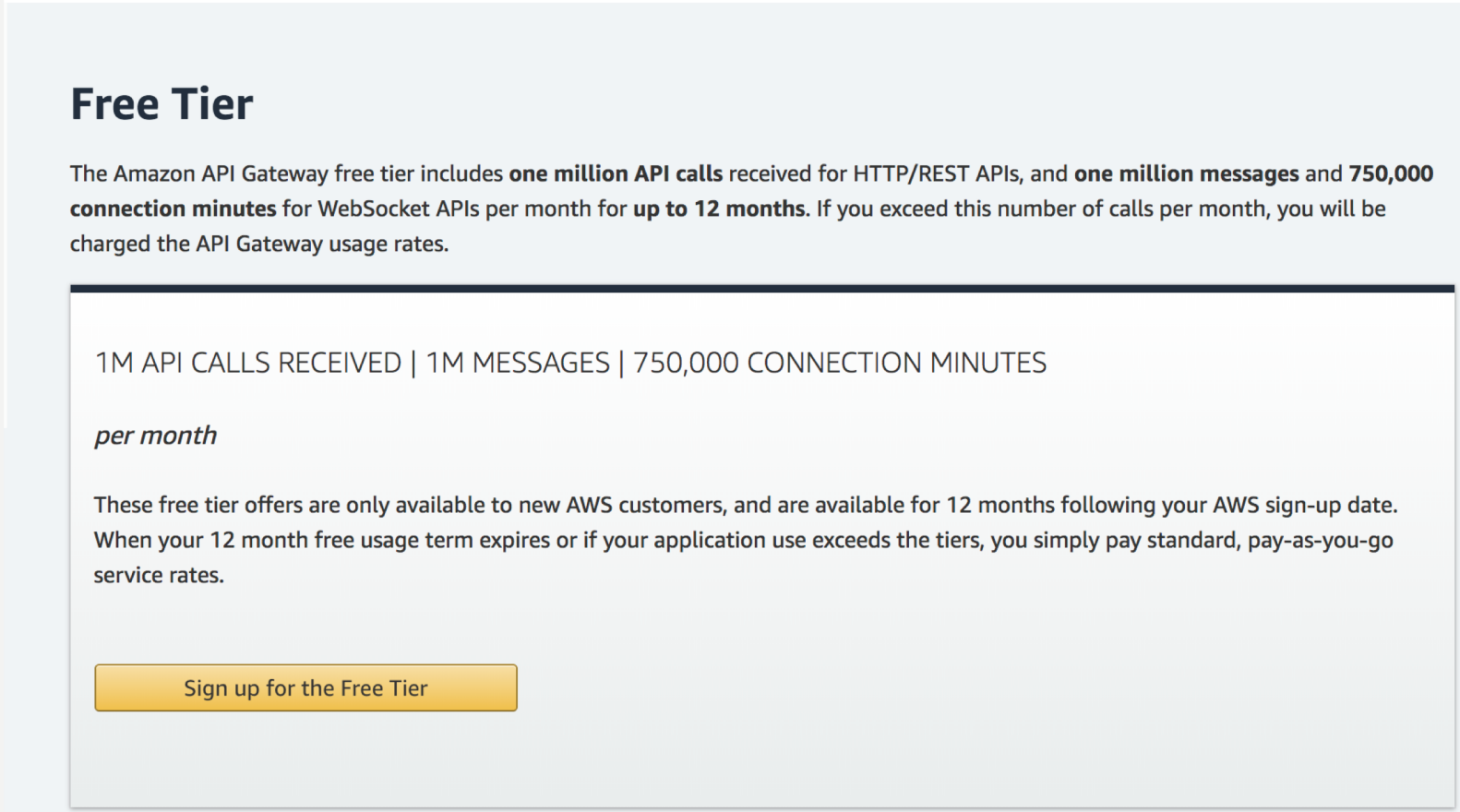
First 400,000 GB-seconds per month, up to 3.2M seconds of compute time, are free.

$0.00001667 FOR EVERY GB-SECOND USED THEREAFTER

The price depends on the amount of memory you allocate to your function.

* I am not liable for any accidental expenditures you might make when using AWS

API GATEWAY (USED TO CALL THE LAMBDA FUNCTION) PRICING



Free Tier

The Amazon API Gateway free tier includes **one million API calls** received for HTTP/REST APIs, and **one million messages** and **750,000 connection minutes** for WebSocket APIs per month for **up to 12 months**. If you exceed this number of calls per month, you will be charged the API Gateway usage rates.

1M API CALLS RECEIVED | 1M MESSAGES | 750,000 CONNECTION MINUTES

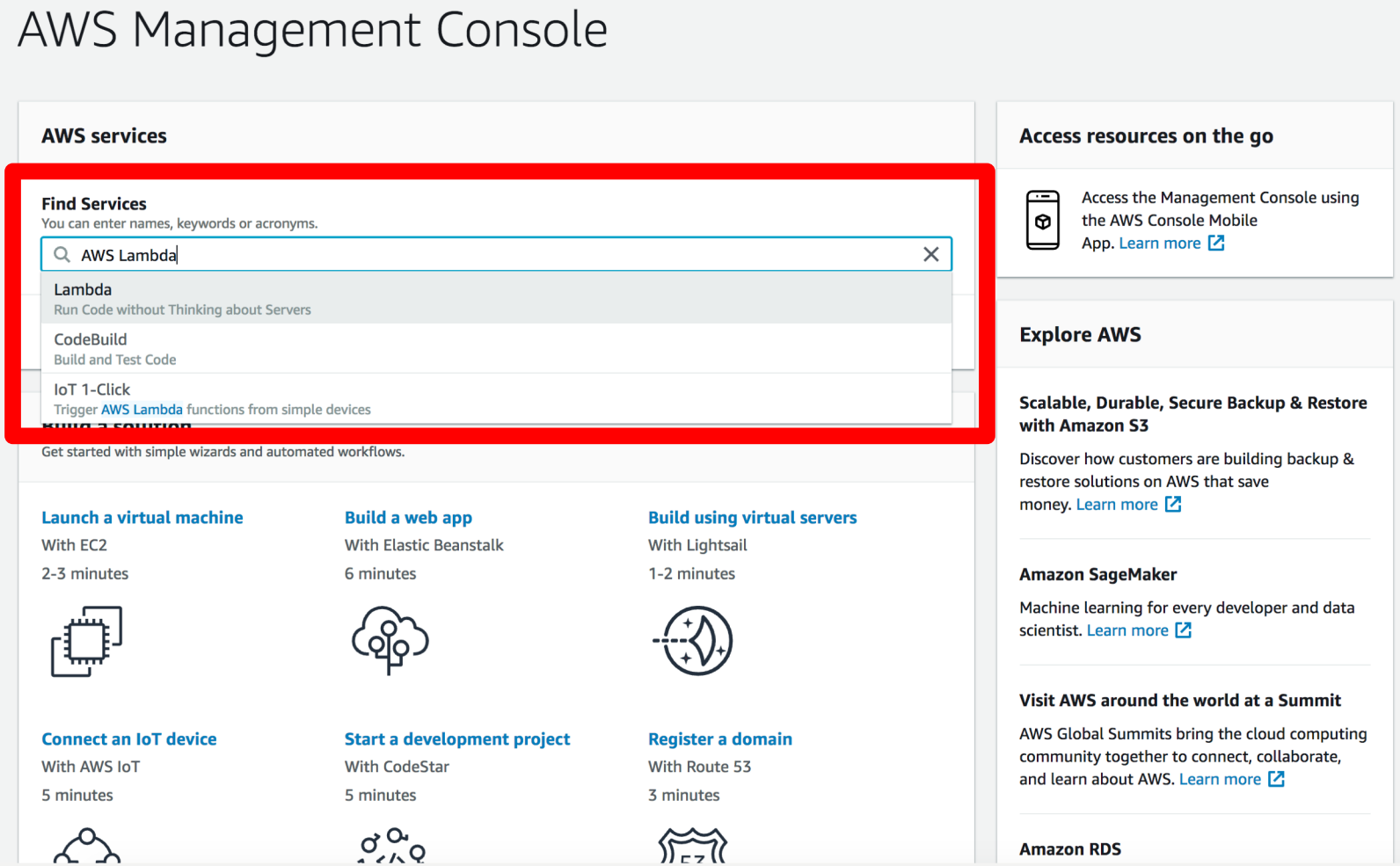
per month

These free tier offers are only available to new AWS customers, and are available for 12 months following your AWS sign-up date. When your 12 month free usage term expires or if your application use exceeds the tiers, you simply pay standard, pay-as-you-go service rates.

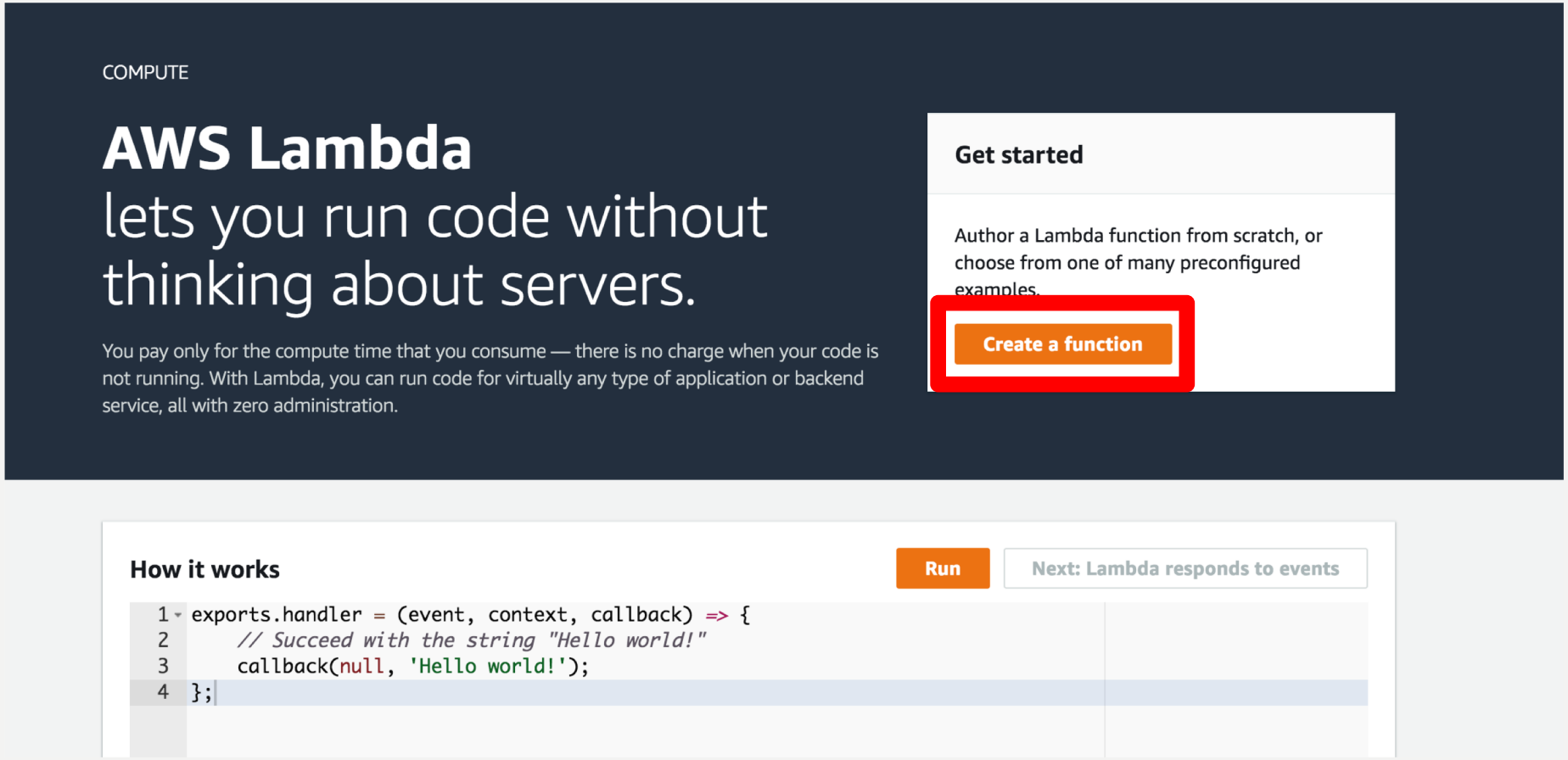
Sign up for the Free Tier

* Free tier expires after12 months of using AWS
* I am not liable for any accidental expenditures you might make when using AWS

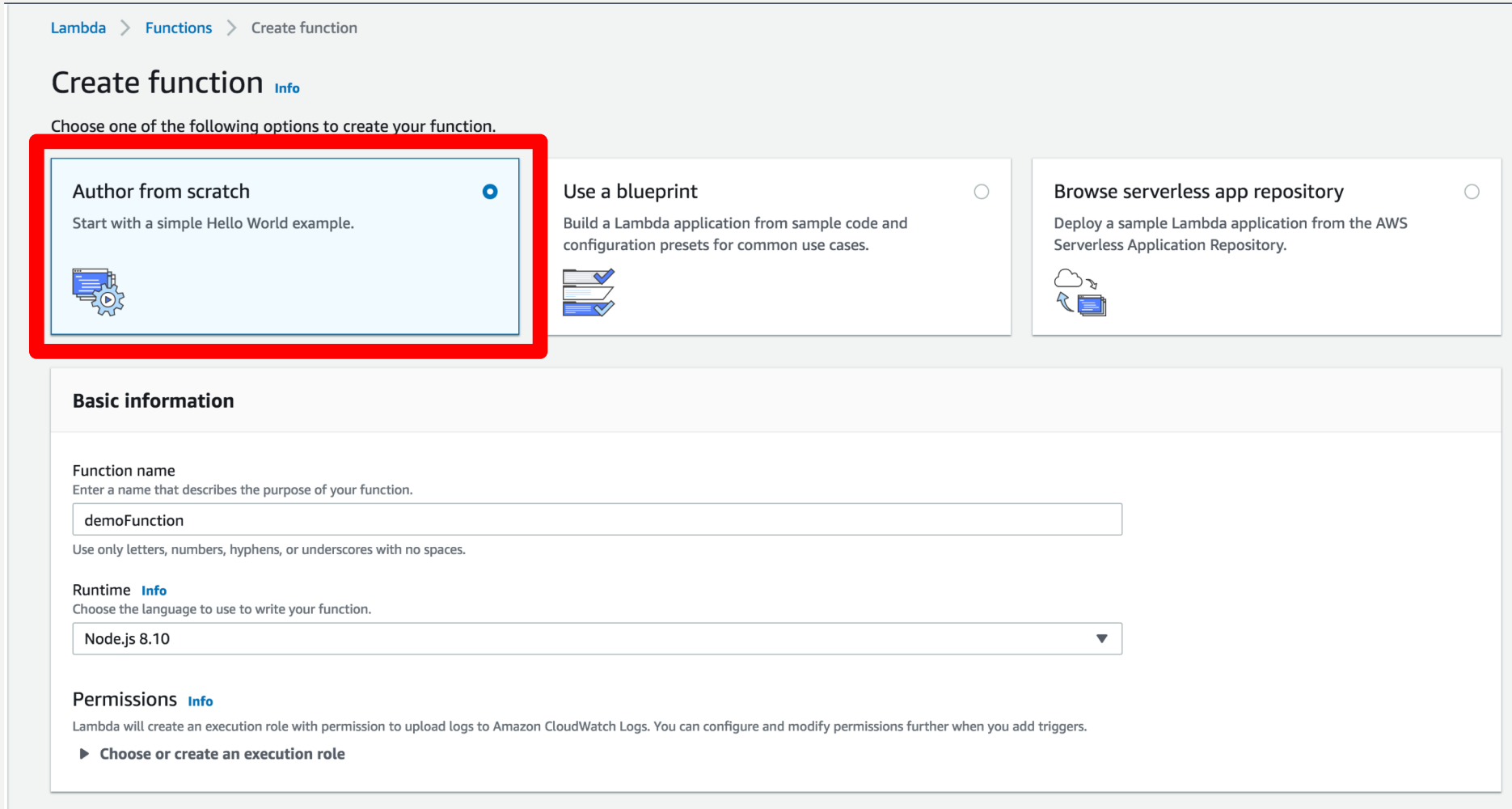
LET'S BEGIN: GO TO THE AWS CONSOLE



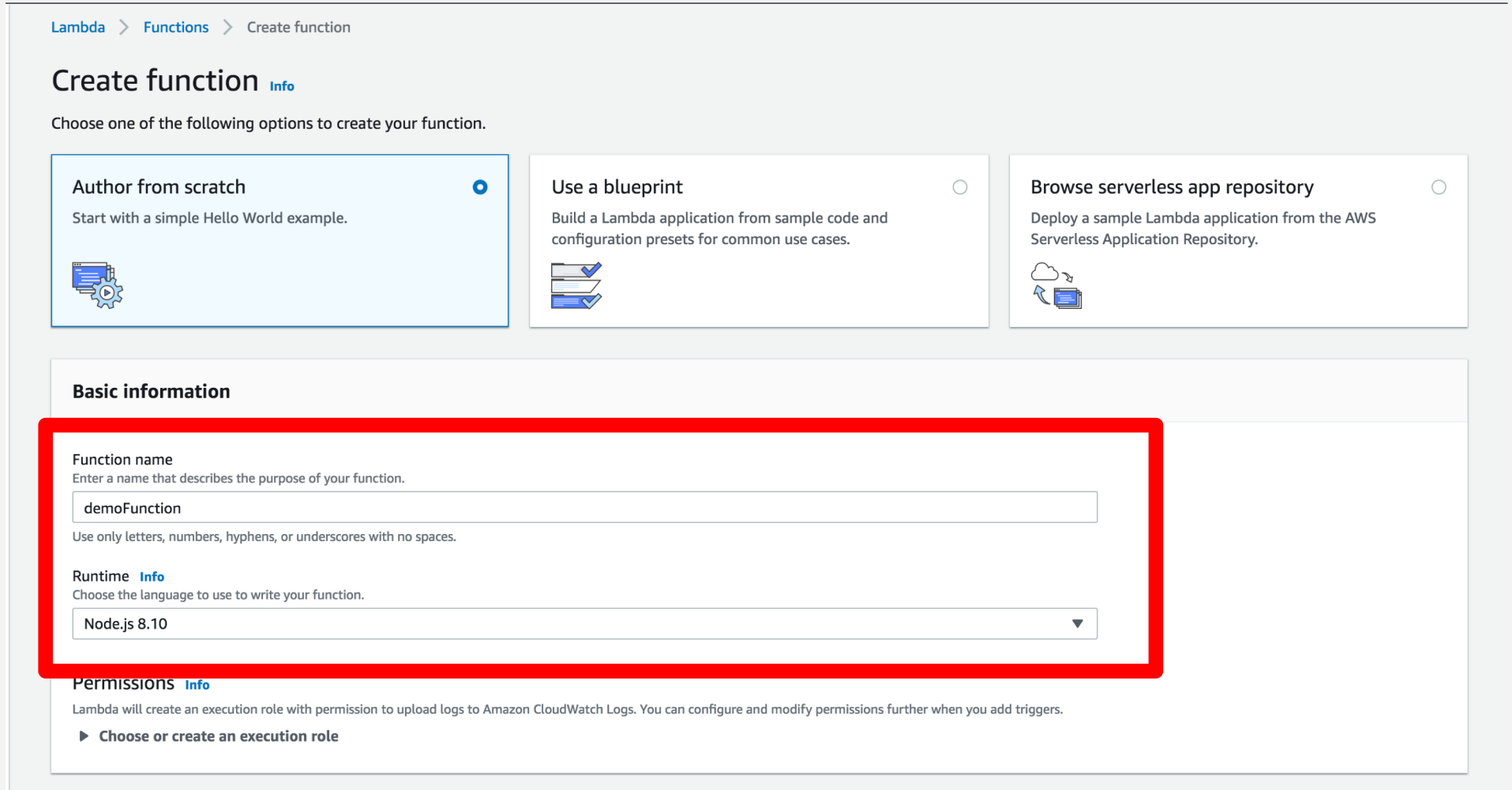
CLICK “CREATE A FUNCTION”



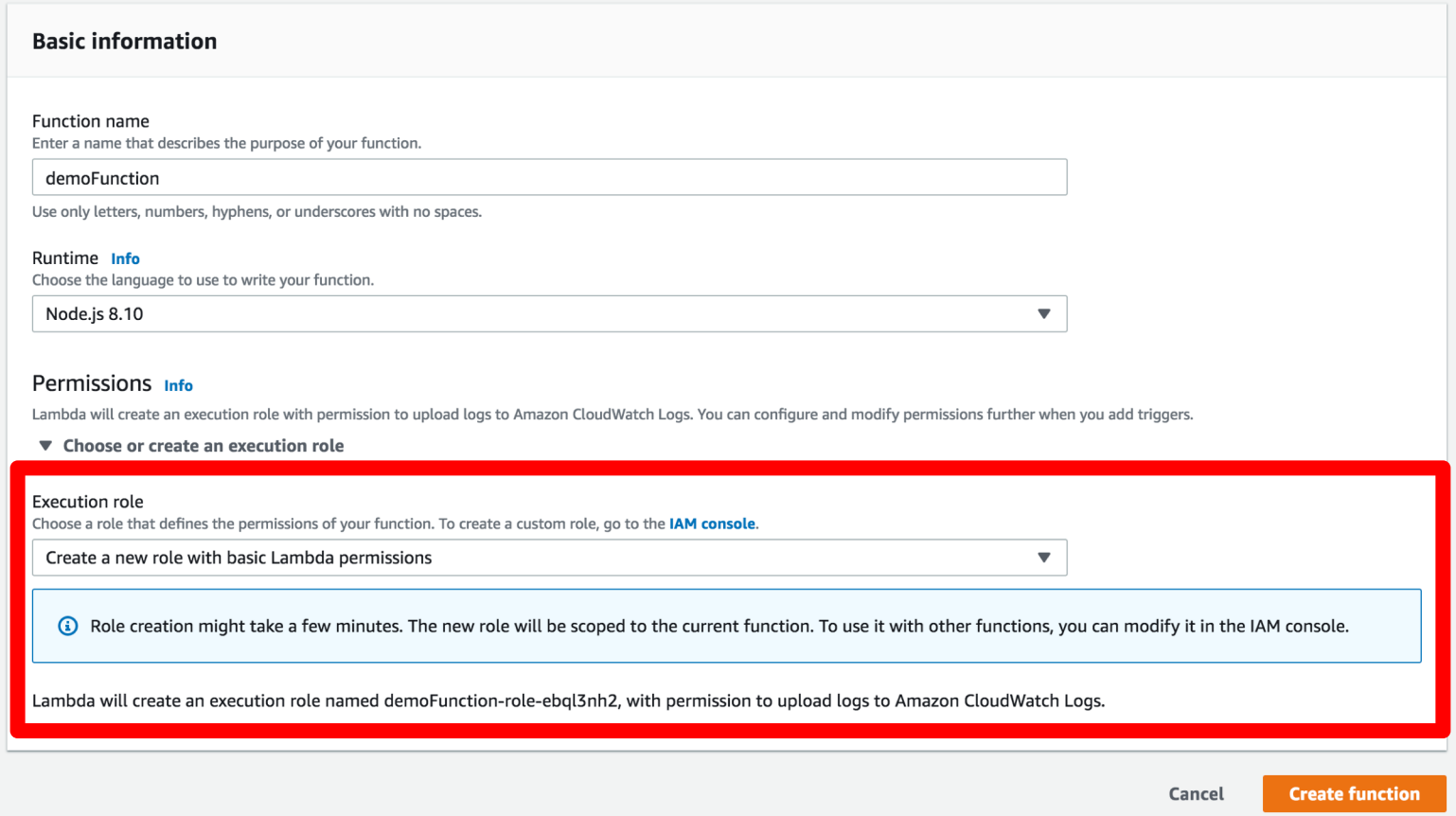
CHOOSE AUTHOR FROM SCRATCH



SET FUNCTION NAME AND CHOOSE NODEJS VERSION

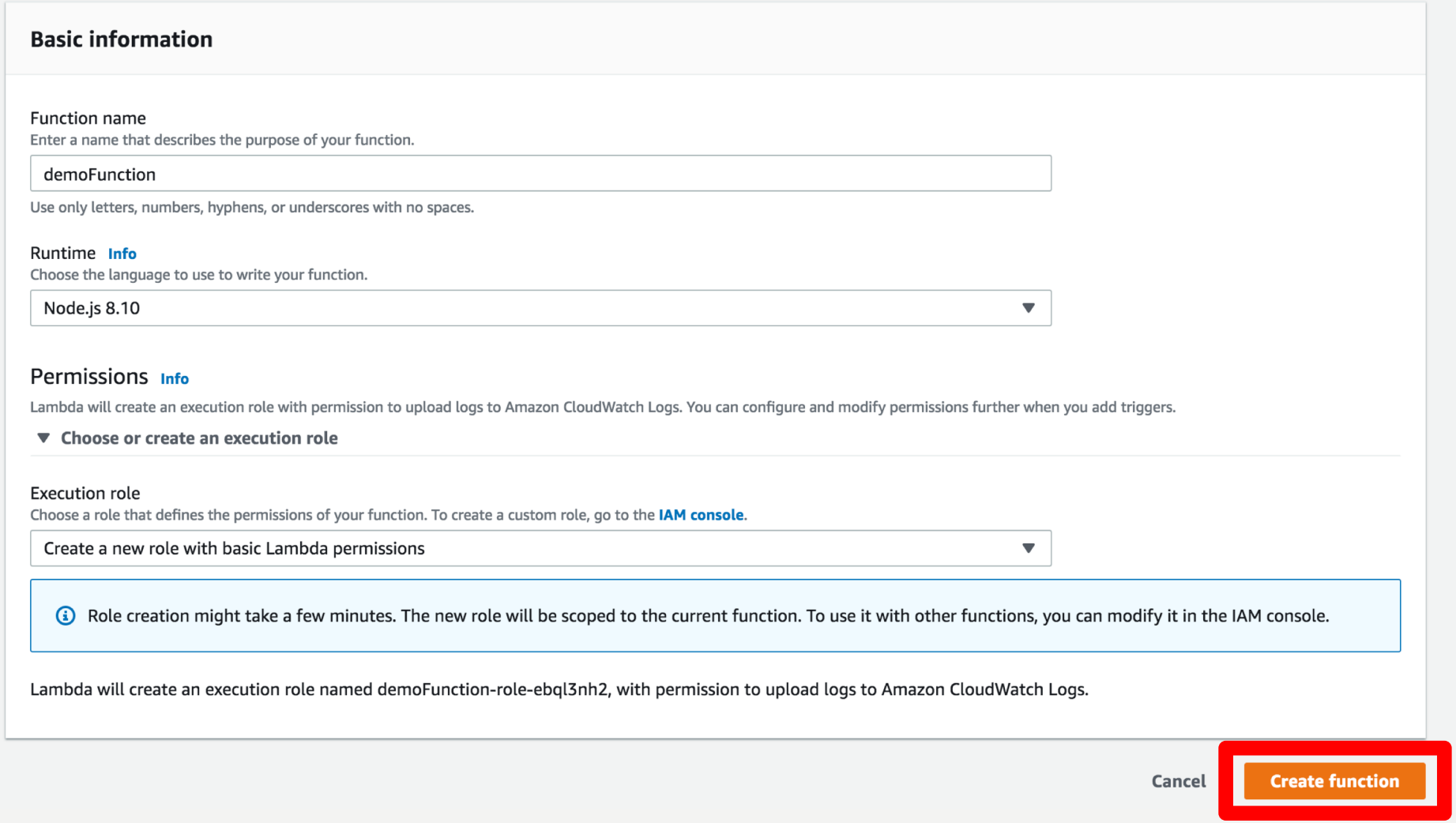


CREATE A NEW ROLE WITH BASIC LAMBDA PERMISSIONS

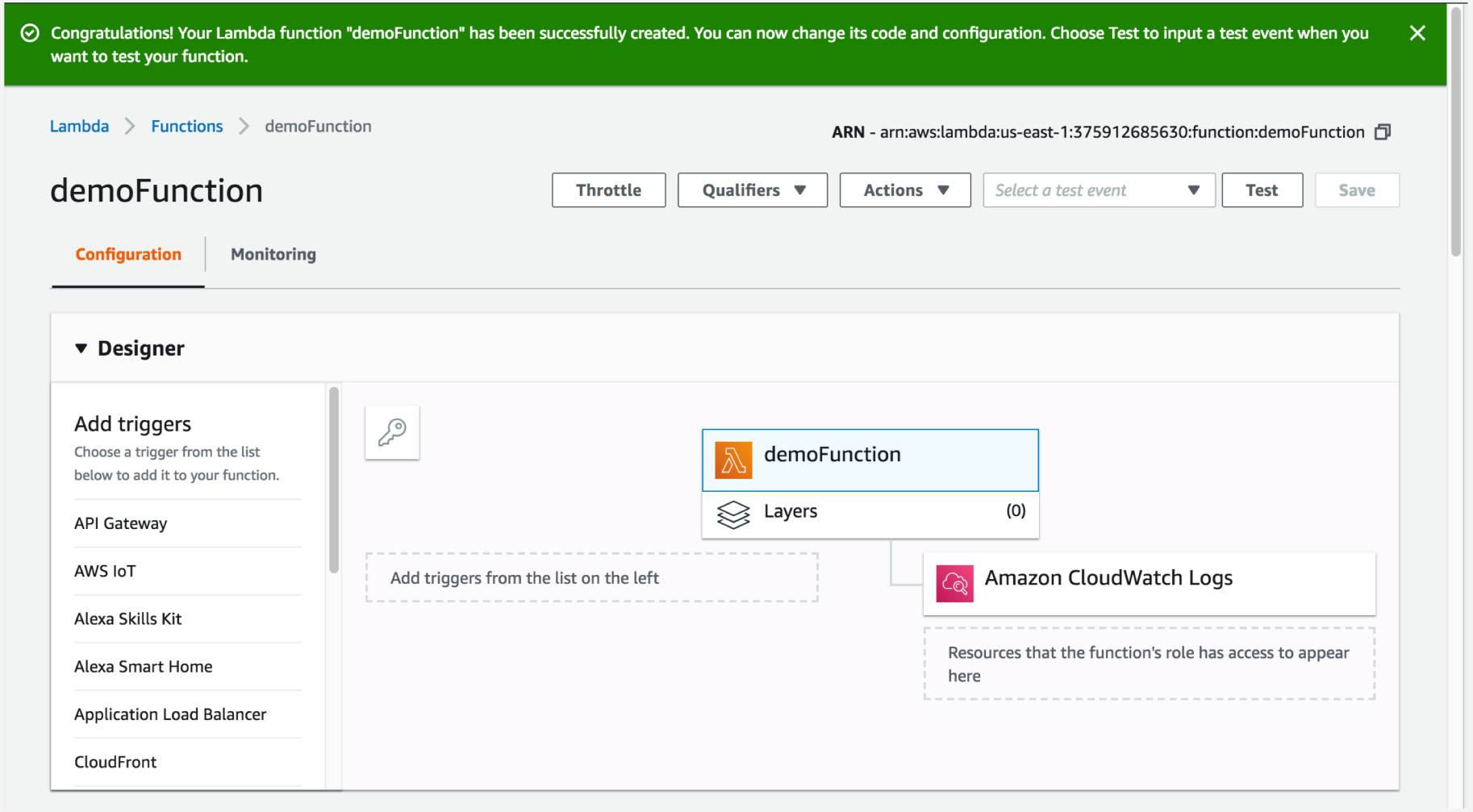


Execution role – is an IAM entity that defines a set of permissions for making AWS service requests

CLICK “CREATE FUNCTION”

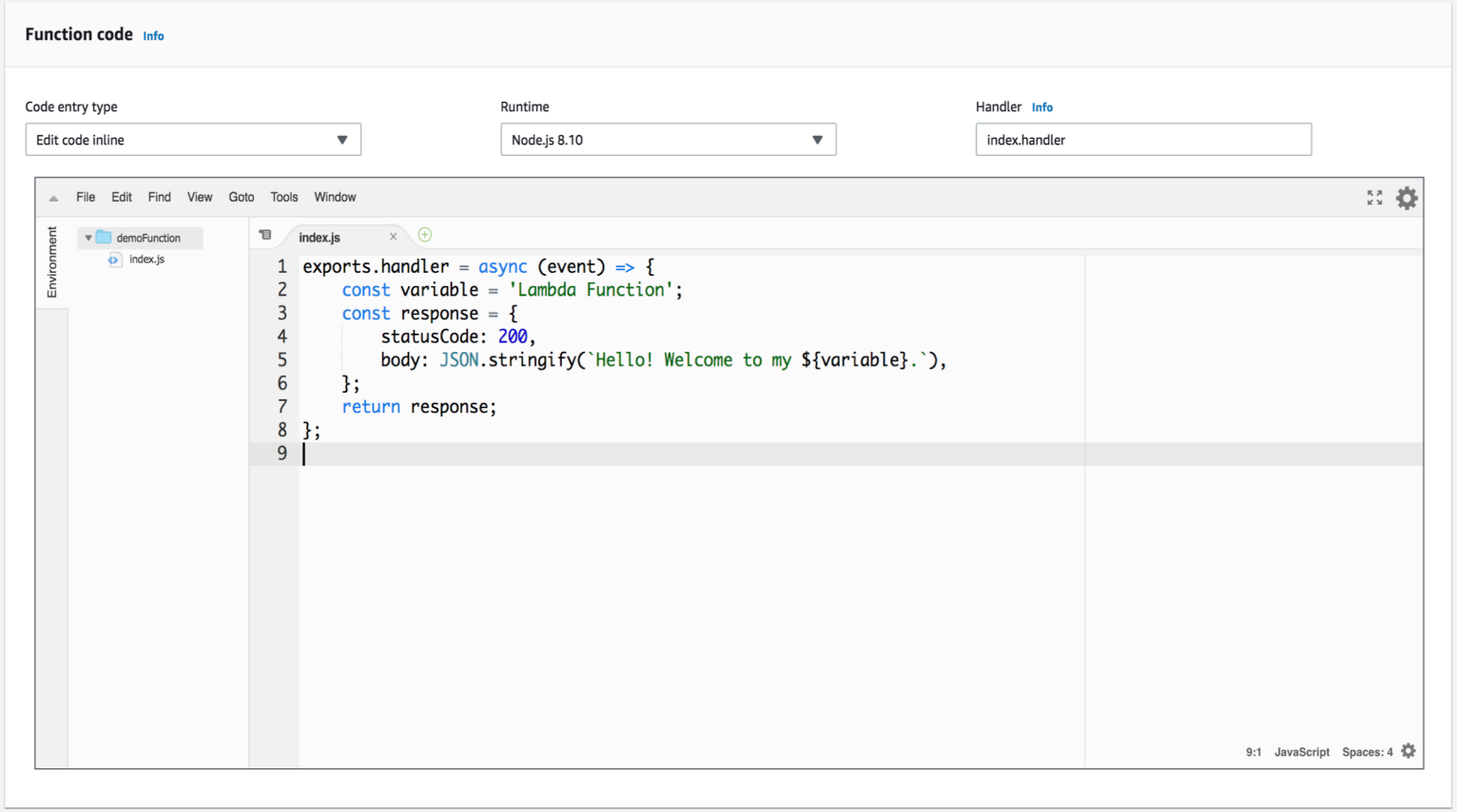


CONGRATS YOU'VE CREATED A LAMBDA FUNCTION! (KIND OF)



SCROLL DOWN TO THE IDE AND LET'S DIVE INTO THE CODE

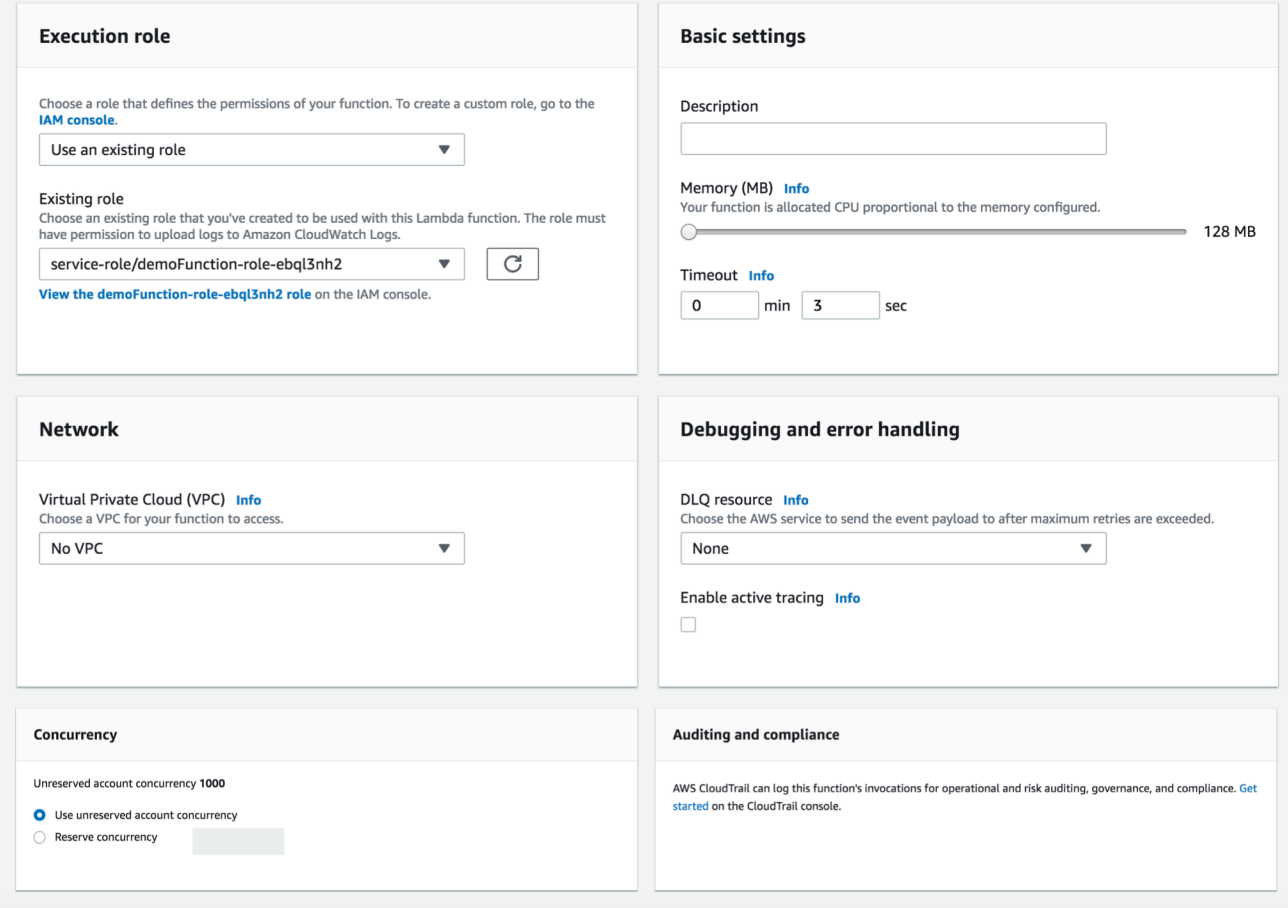
- To the right, we have a bit of Node code
- AWS Lambda invokes your Lambda function via a handler object.
  A handler represents the name of your Lambda function and serves as the entry point that AWS Lambda uses to execute your function code.
- Using "async" means that this function is not going to hold up any other code and will run when triggered
- When the function is triggered, we return a JSON response with a status of 200 – OK



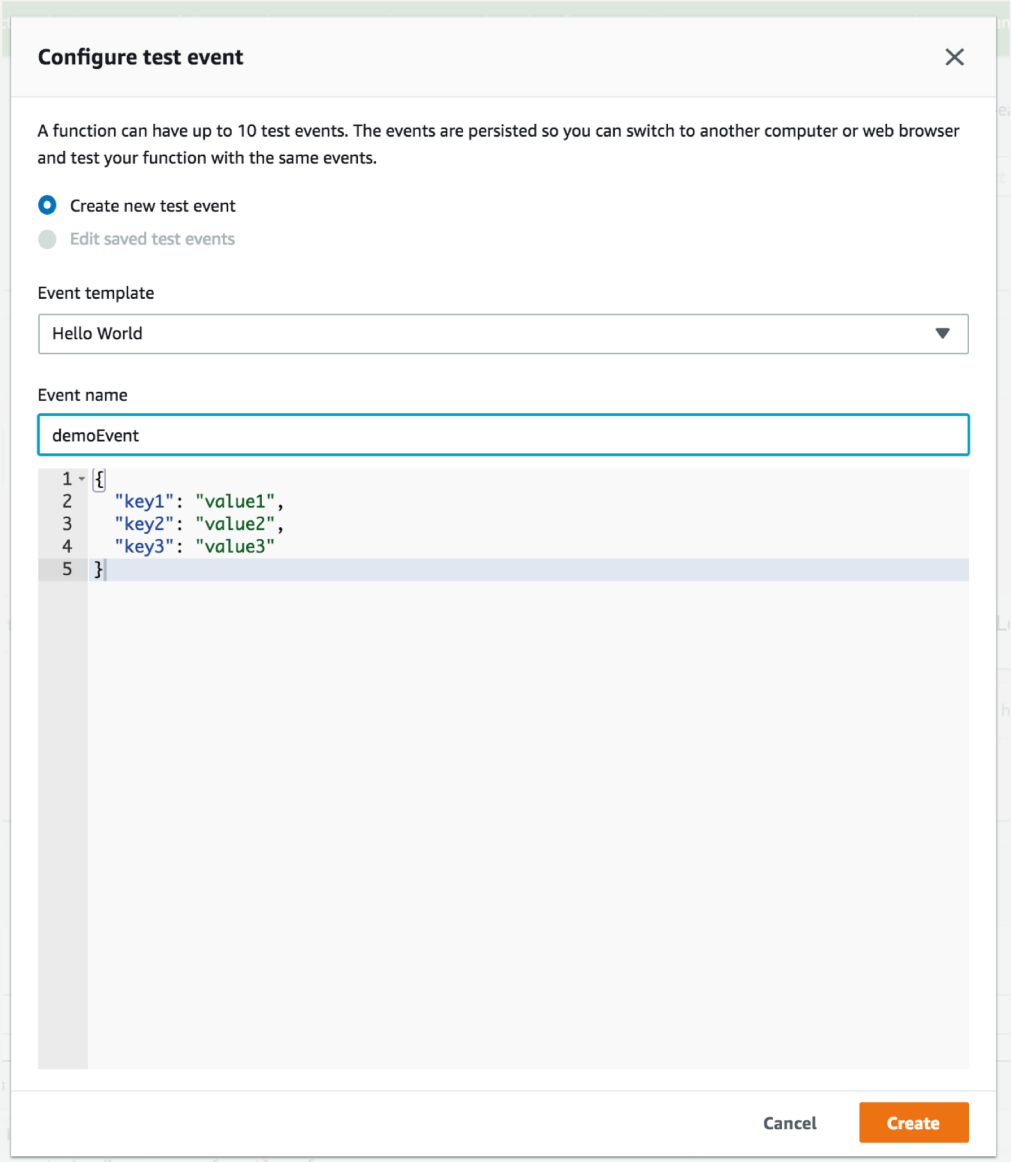
KEEP THE SETTINGS AS IS (128MS AND 3 SECOND TIMEOUT)

Timeout – You pay for the AWS resources that are used to run your Lambda function. To prevent your Lambda function from running indefinitely, you specify a timeout. When the specified timeout is reached, AWS Lambda terminates execution of your Lambda function.

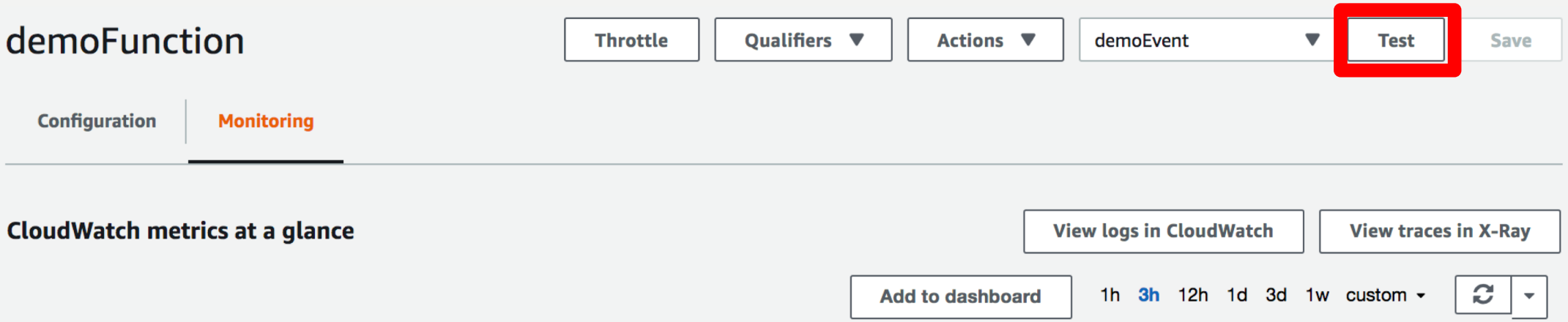
Concurrency - The concurrency limit determines how many function invocations can run simultaneously in one region. The limit applies to all functions in the same region and is set to 1000 by default.



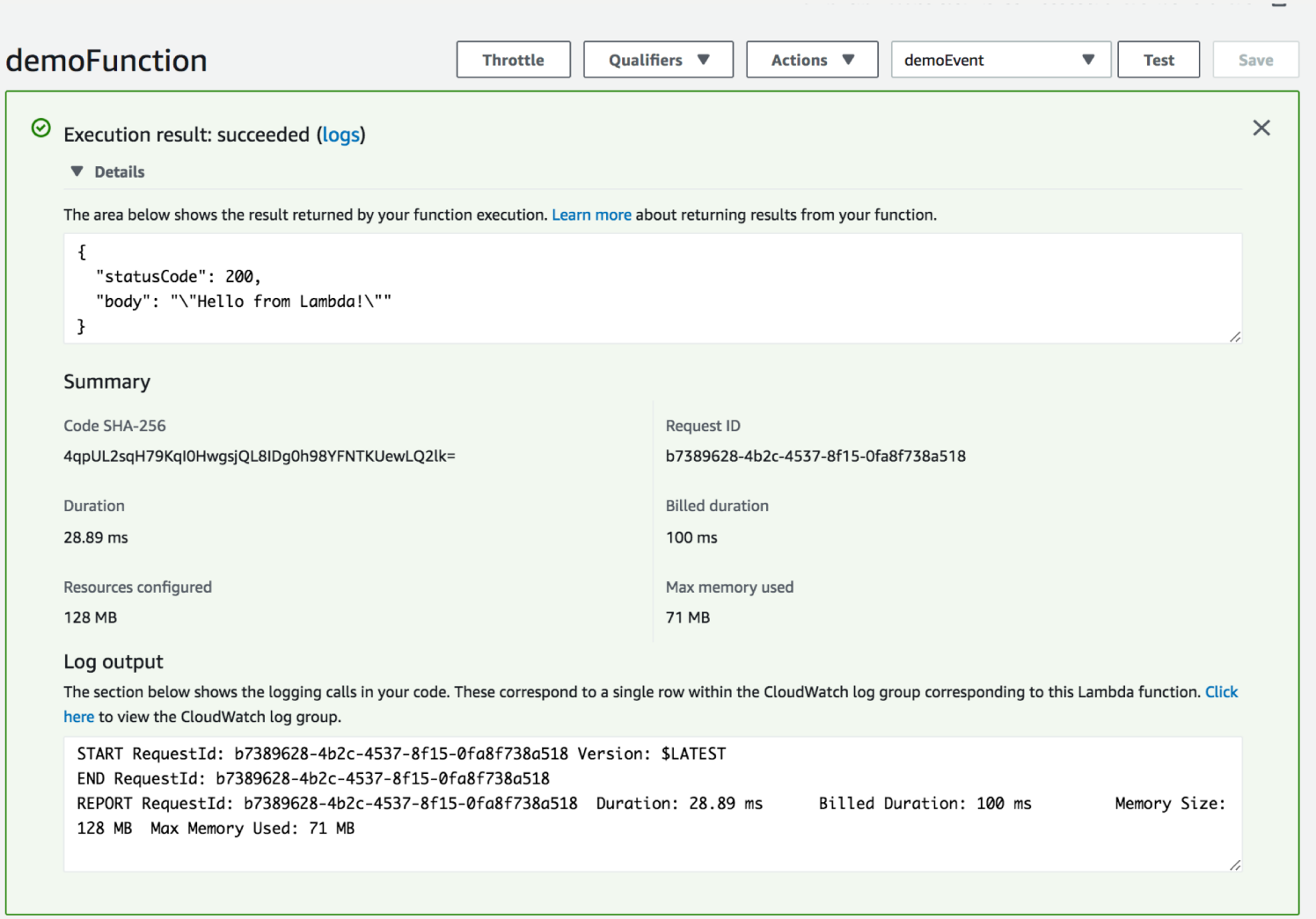
CREATE AN EVENT TO TEST YOUR FUNCTION



CLICK ON THE "TEST" BUTTON NEXT TO YOUR SELECTED EVENT

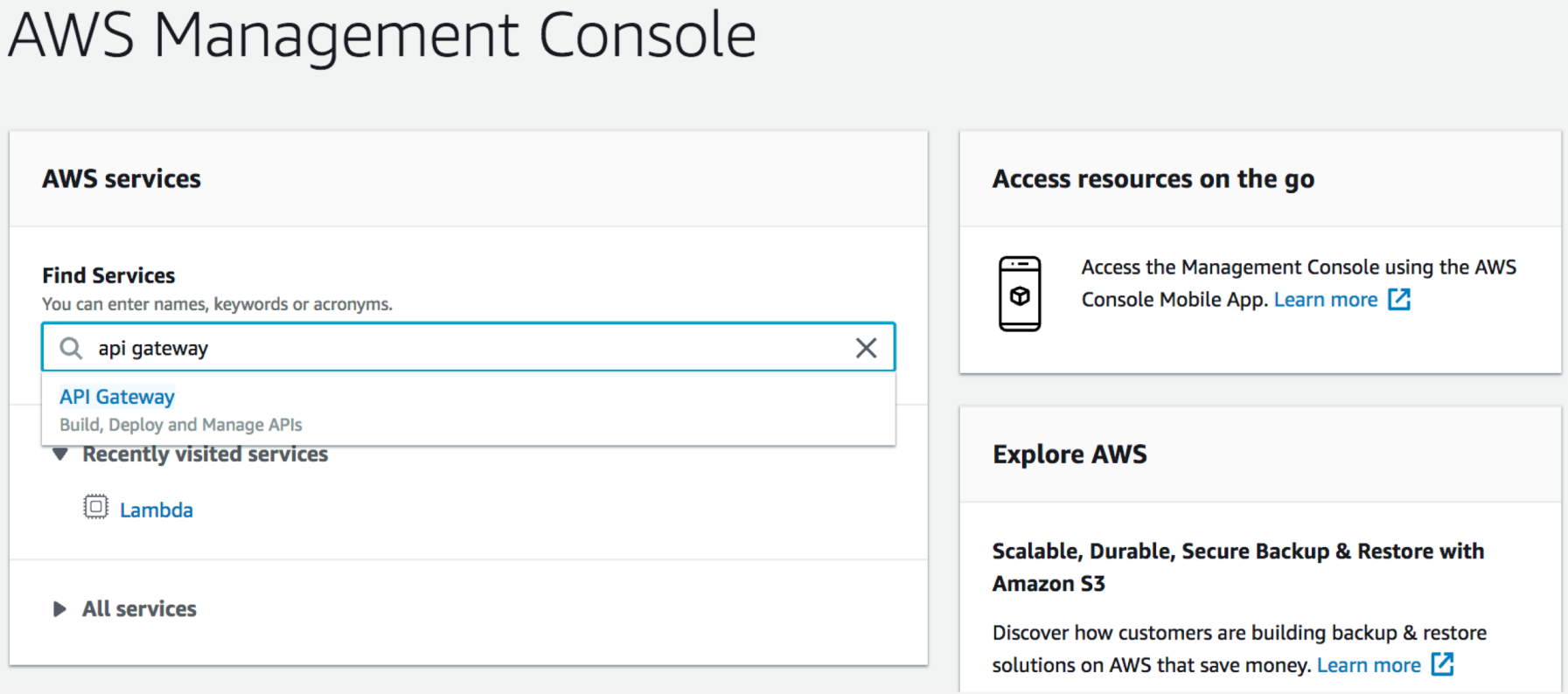


VIEW YOUR SERVERLESS FUNCTION RESULTS



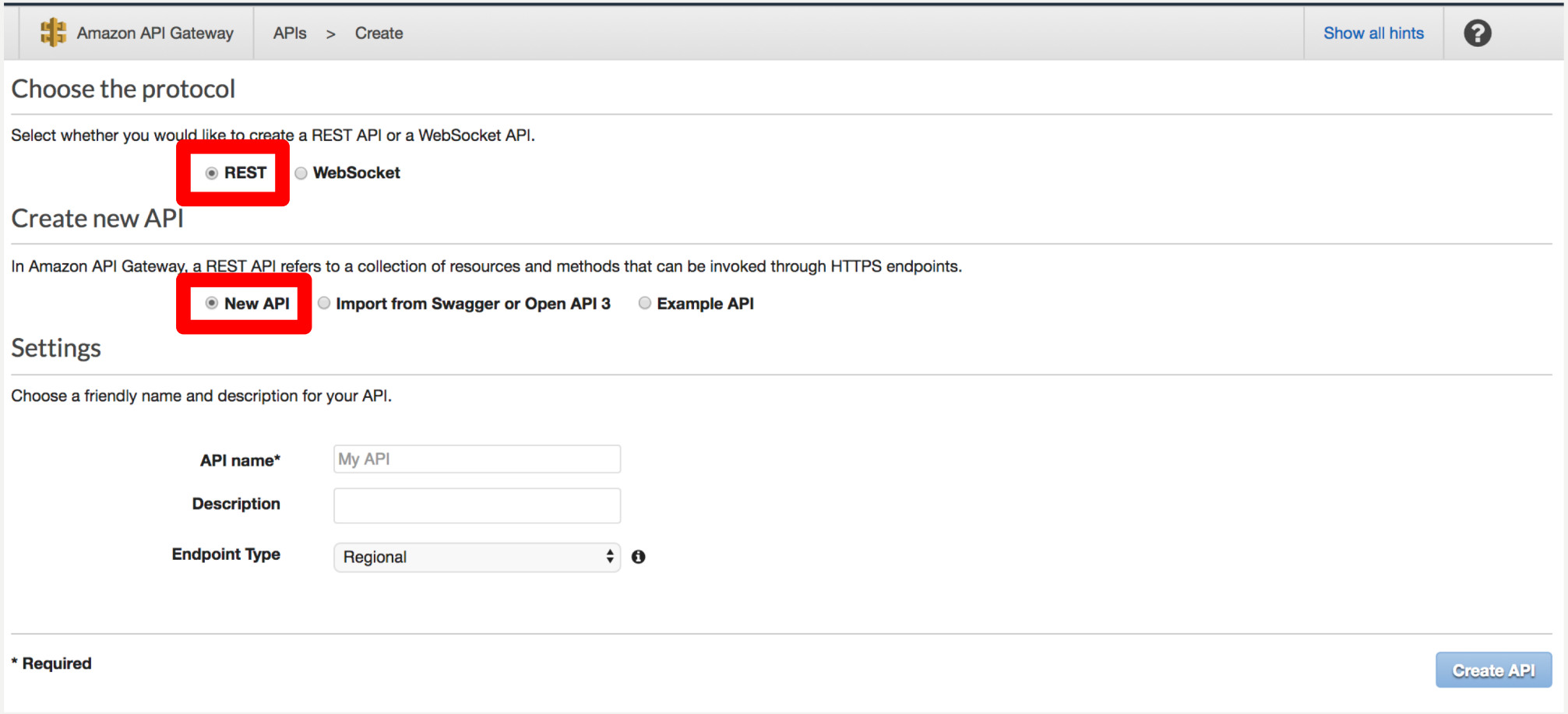
TRIGGERING YOUR FUNCTION WITH A REST CALL

Go back to the AWS console and search for and click on API Gateway

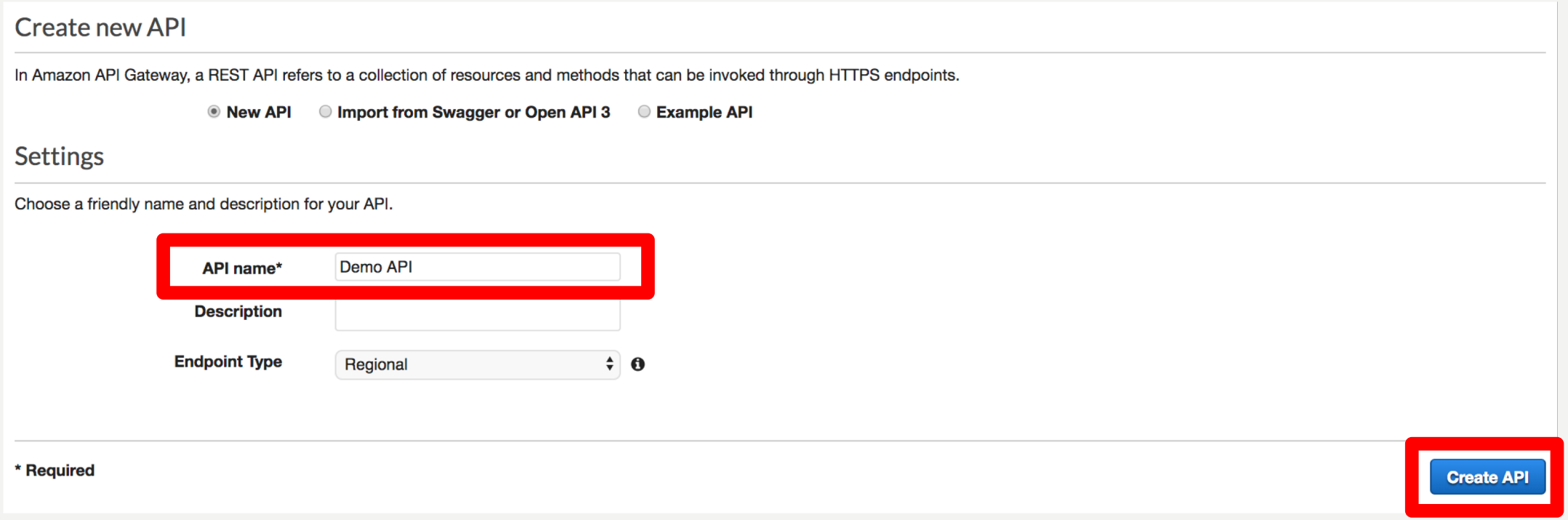


CLICK GET STARTED ON THE API GATEWAY HOME PAGE

CHOOSE THE REST OPTION AND CREATE A "NEW API"

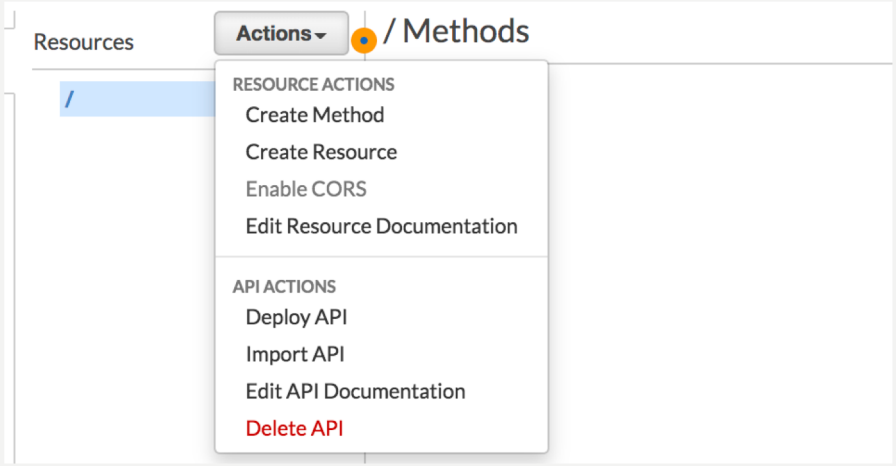


ENTER IN YOUR API NAME AND CLICK "CREATE API"

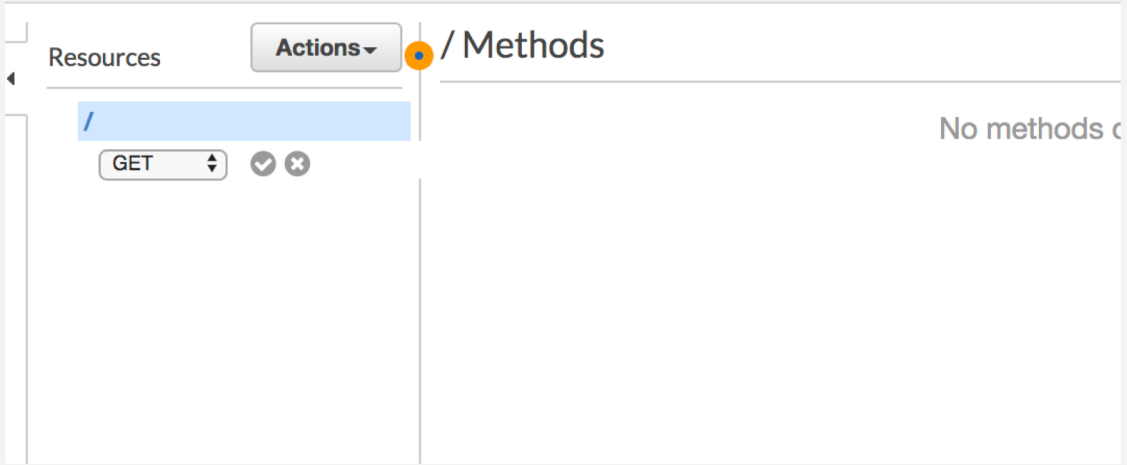


LET'S MAKE A "GET" REQUEST

Step #1: Click Create Method



Step #2: Click on the "GET" method and check button



Step #3: Set the Integration type to "Lambda Function"

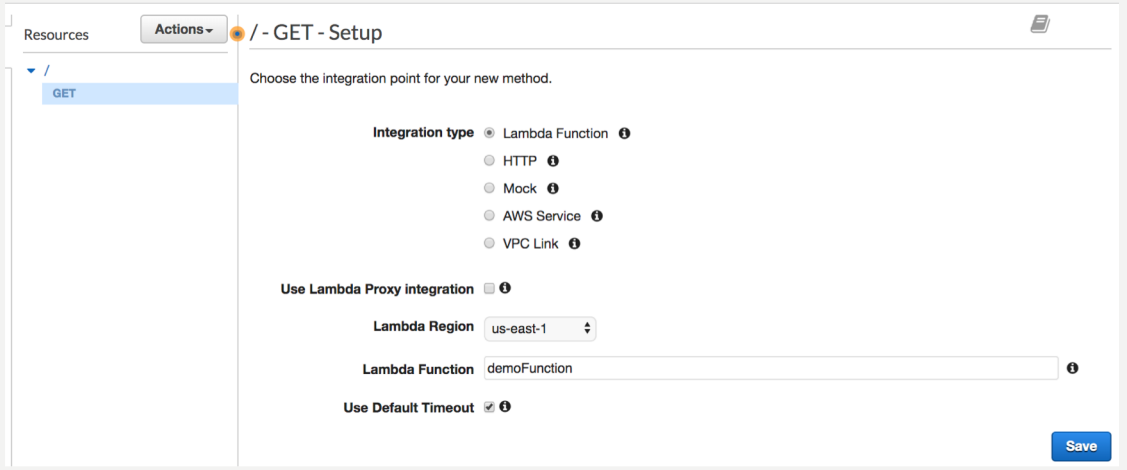
Step #4: Keep Lambda Proxy Integration empty (we will not be using a proxy)

Step #5: Set Lambda region to the region where your users will be located (us-east-1 is fine)

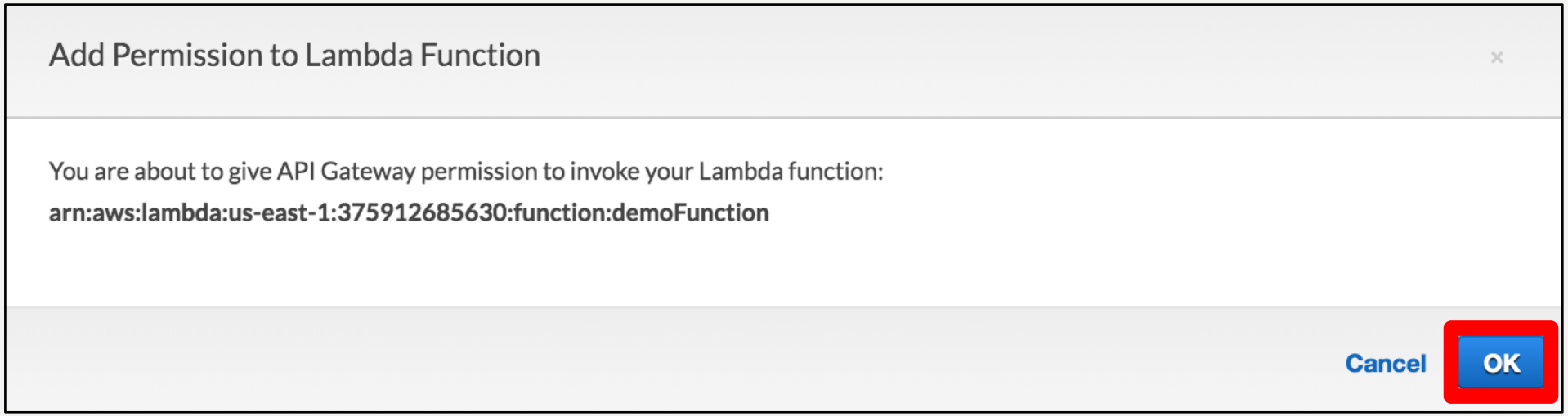
Step #6: Set function to your function name

Step #7: Use default timeout

Step #8: Click save

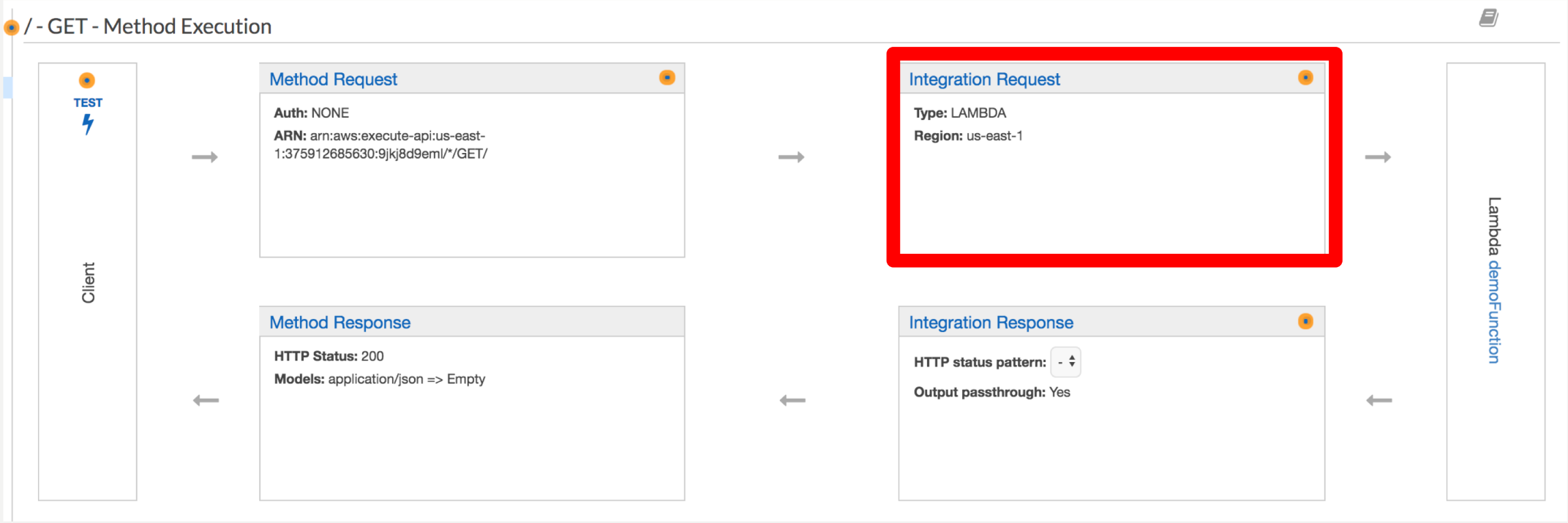


GIVE PERMISSION TO API GATEWAY TO CALL YOUR LAMBDA FUNCTION



SET UP THE INTEGRATION REQUEST

Click on the Integration Response configuration tile



SET UP THE INTEGRATION REQUEST

The Integration Request should be set to the following values:

Type: Lambda Function

Proxy Integration: False

Lambda Region: (the region you used for your Lambda function)

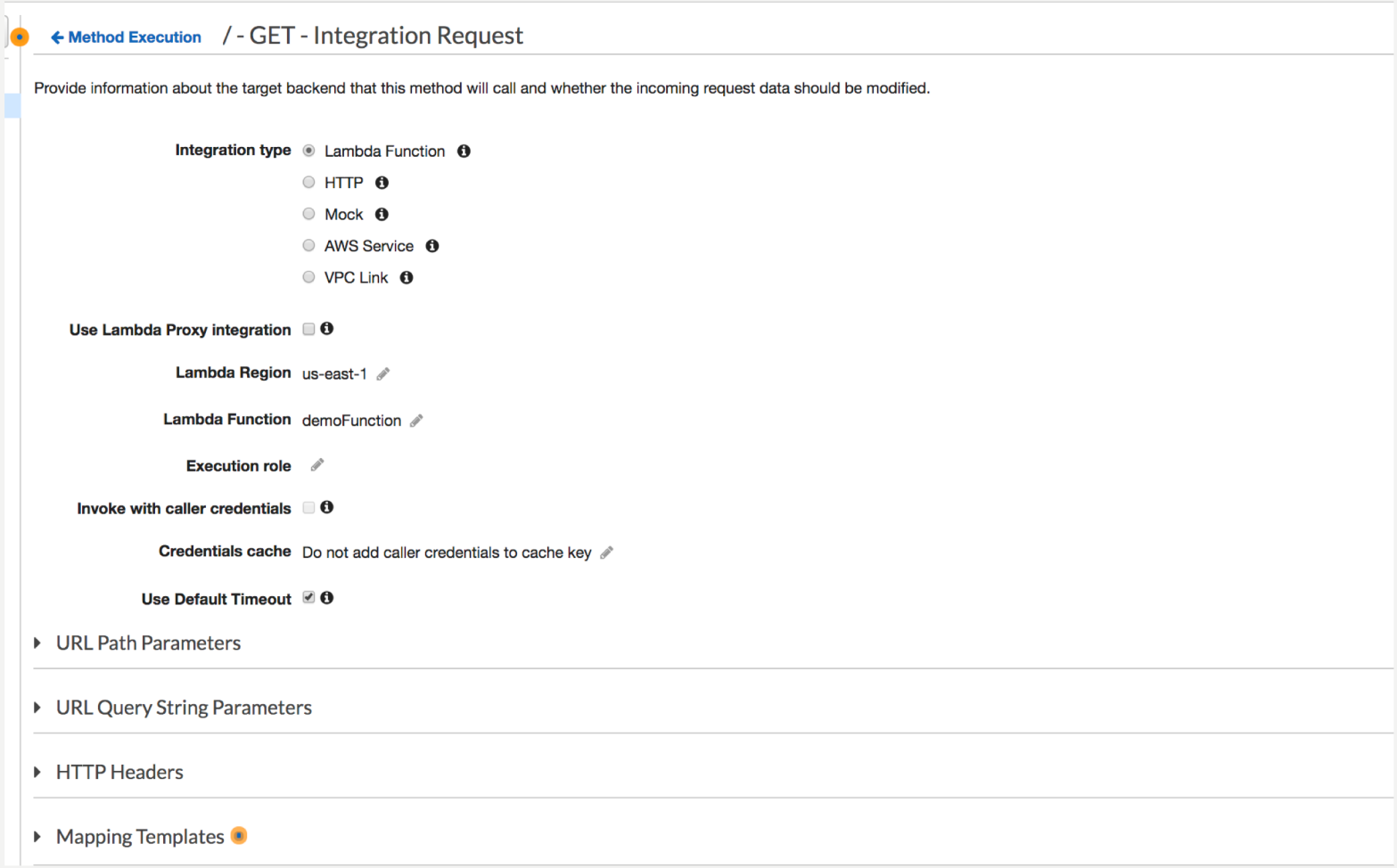
Lambda Function: (Function name)

Execution role: (empty)

Caller Credentials: False

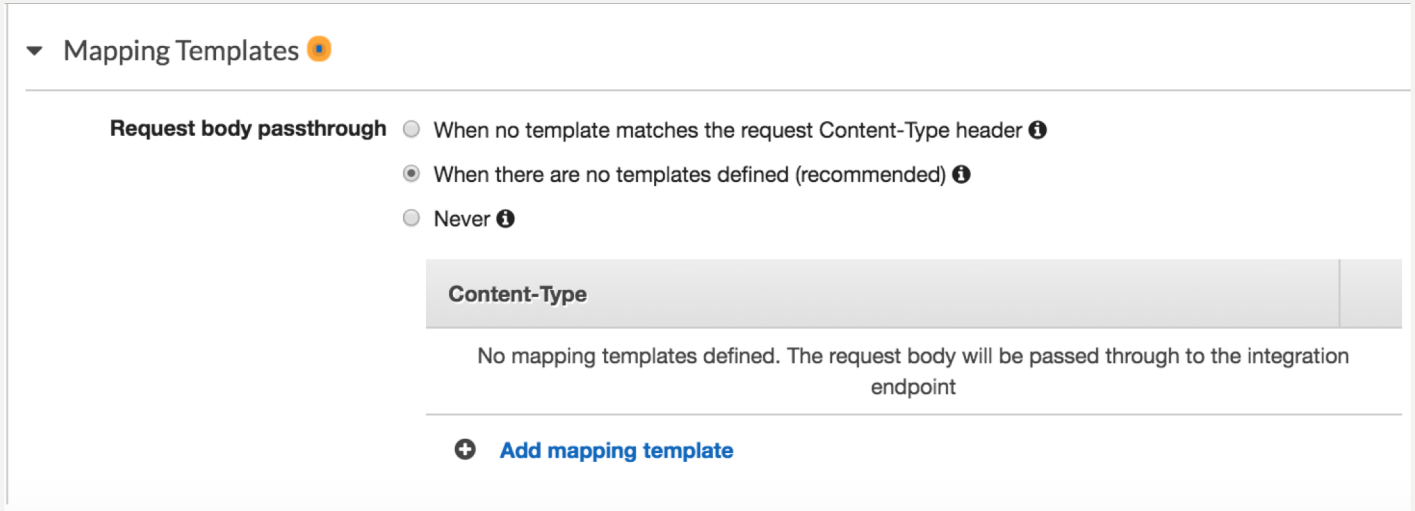
Credentials Cache: (empty)

Default Timeout: True



ADD MAPPING TEMPLATE

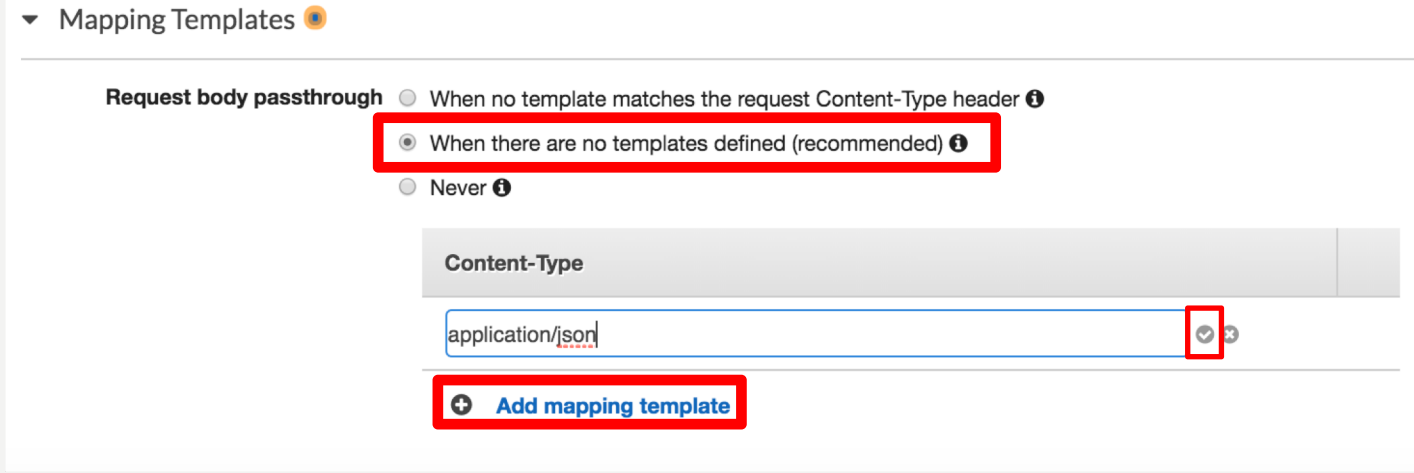
Set the "Requested body passthrough" to: "When there are no templates defined (recommended)"



Click on "Add mapping template" and enter in: "application/json"

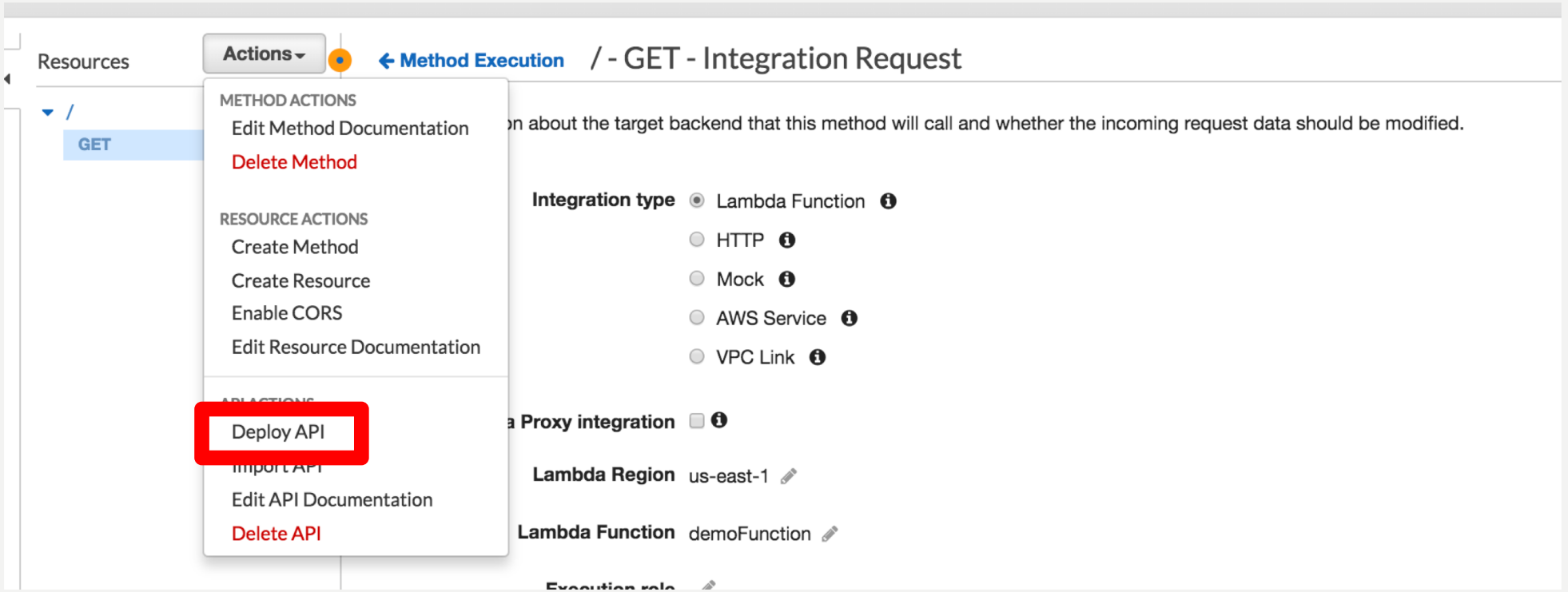
Then click on the check mark next to the input field

Click "Save"

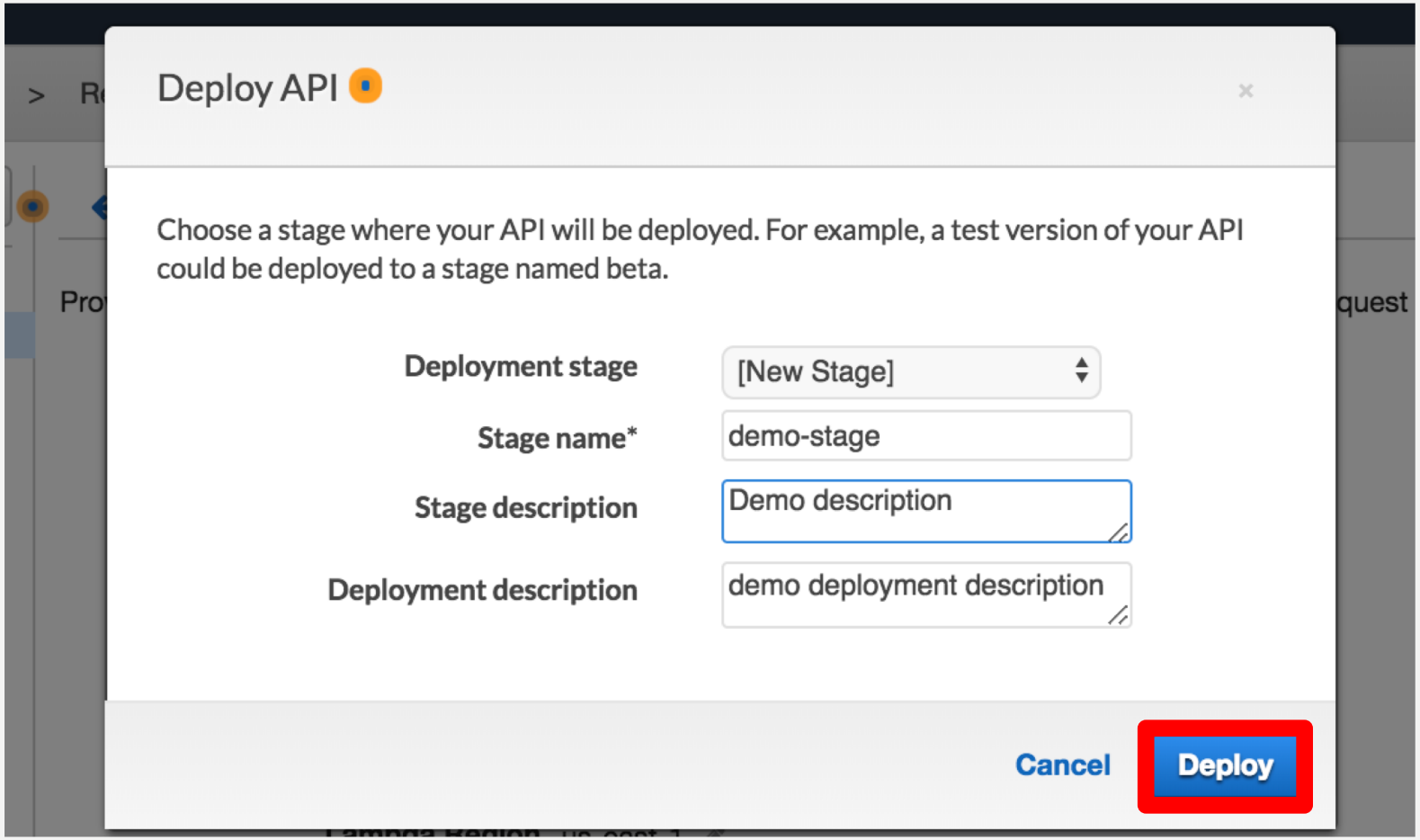


DEPLOY YOUR API!

Click on Deploy API on the Actions Dropdown

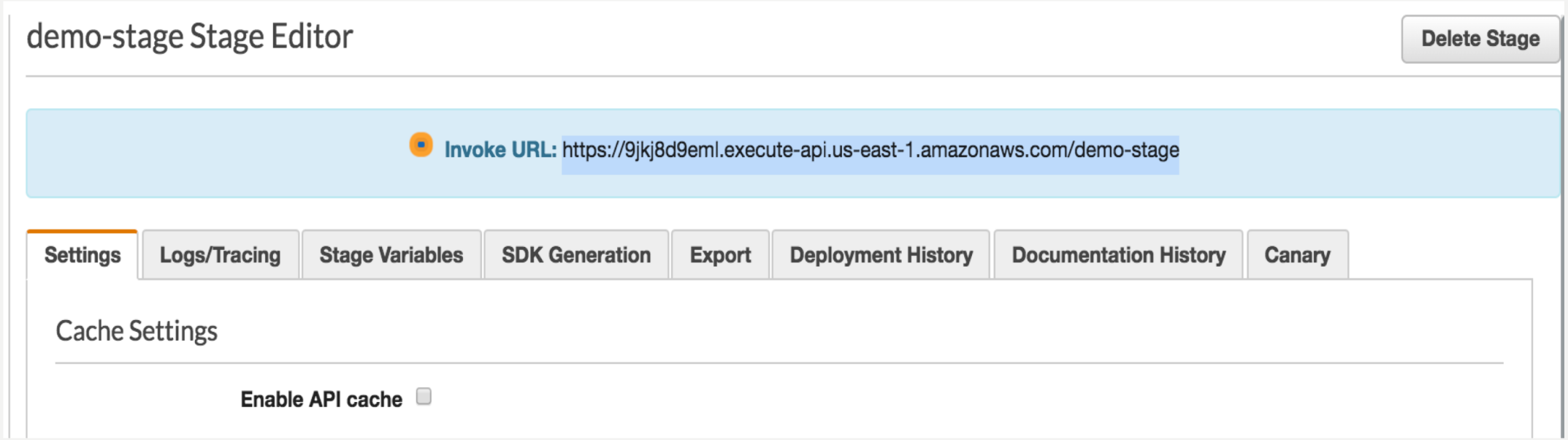


CREATE “NEW STAGE” WHERE YOUR API WILL BE DEPLOYED AND CLICK “DEPLOY”



CLICK ON YOUR "INVOKE URL"

The Invoke URL will be used to call your function.



VIEW YOUR FUNCTION RESULTS

- Your results will be in JSON format which is what we selected for our integration request
- The status code of "200" stands for OKAY and means that our call was successful
- The body of the JSON shows the returned value from the function


```
{
    statusCode: 200,
    body: ""Hello! Welcome to my Lambda Function.""
}
```


*Hint: to more easily read your JSON response, download the JSONView Chrome plugin which auto-formats any JSON that appears in the body (I'm using the plugin to display the above)

CONGRATS ON CREATING YOUR FIRST SERVERLESS FUNCTION!

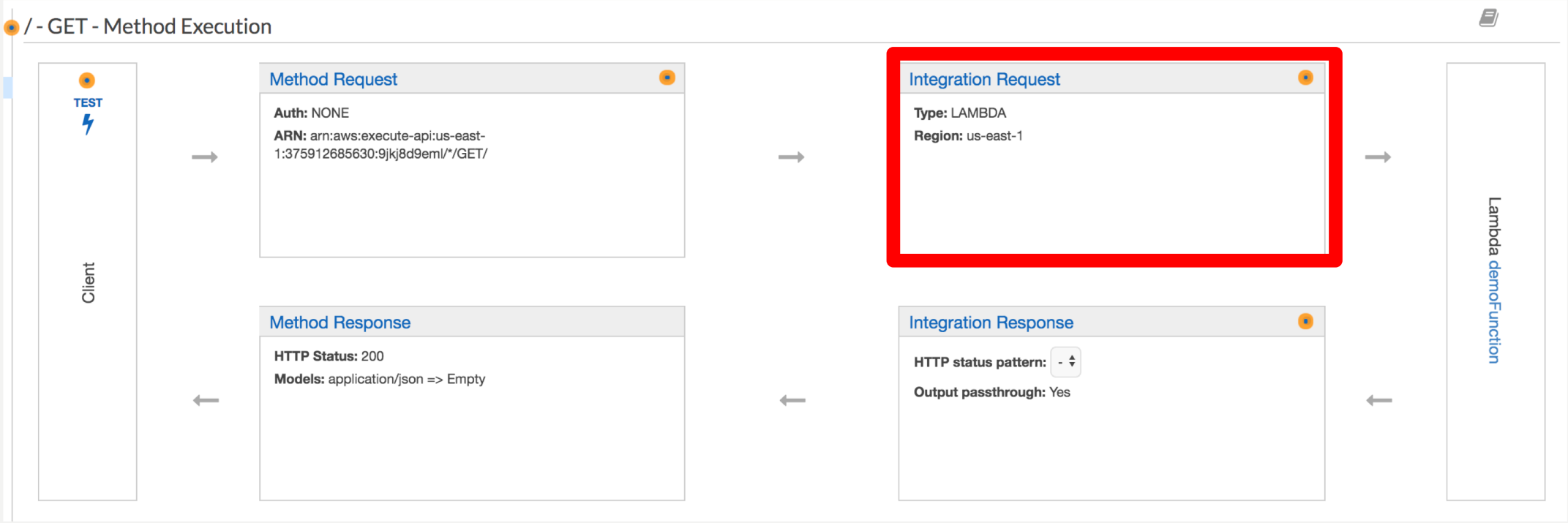
Let's try passing data into your Lambda Function.

To start, go back to API Gateway home: https://console.aws.amazon.com/apigateway/home

Then, click on your newly created API and then the GET request

SET UP THE INTEGRATION REQUEST

Click on the Integration Response configuration tile



SET UP THE INTEGRATION REQUEST

The Integration Request should be set to the following values (These values should already be set from the initial configurtation):

Type: Lambda Function

Proxy Integration: False

Lambda Region: (the region you used for your Lambda function)

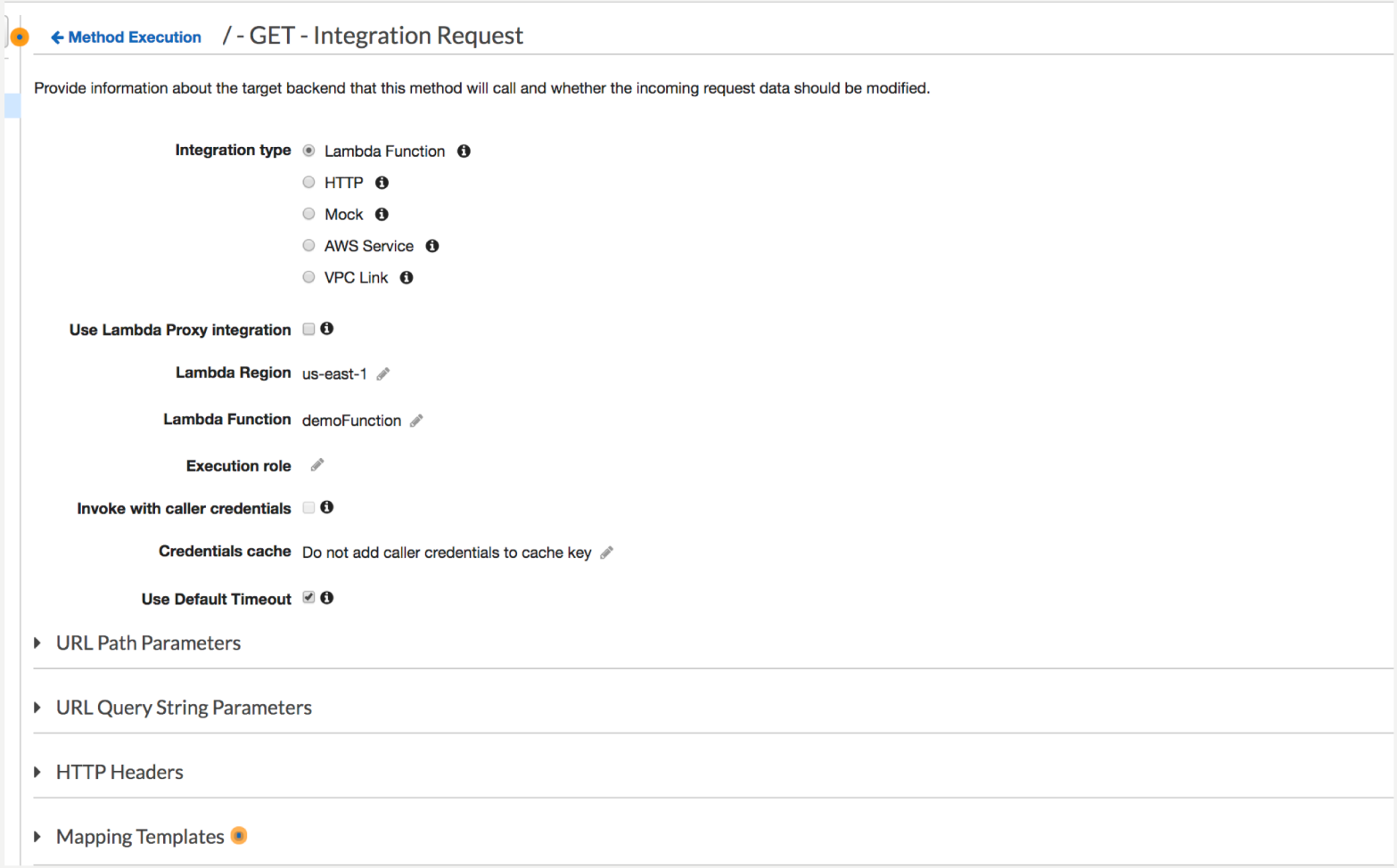
Lambda Function: (Function name)

Execution role: (empty)

Caller Credentials: False

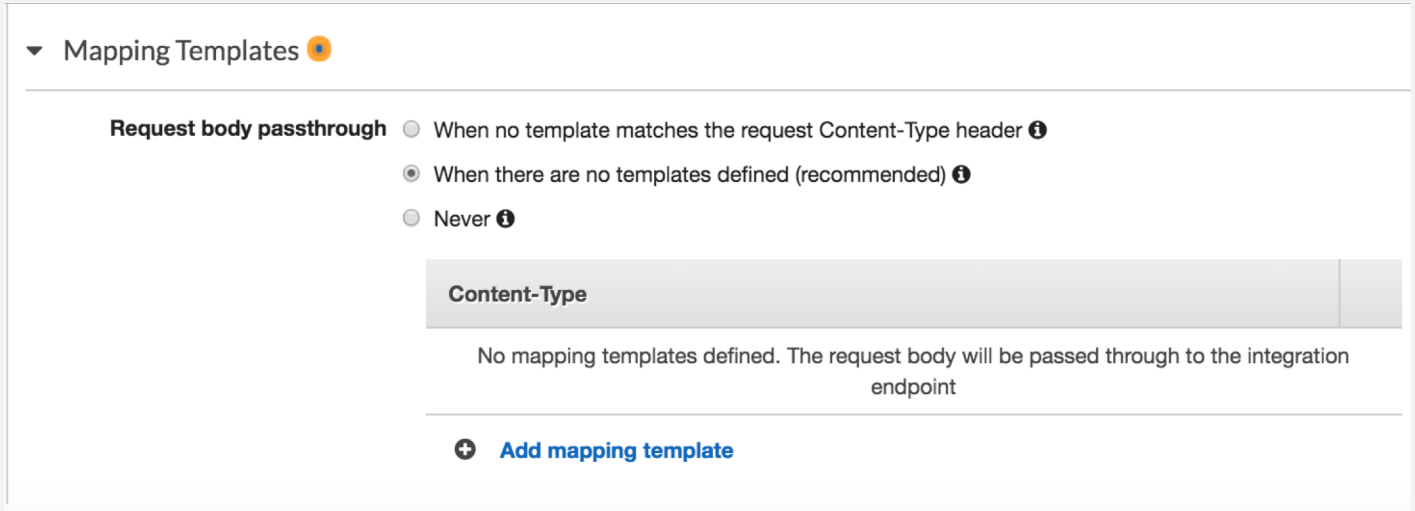
Credentials Cache: (empty)

Default Timeout: True



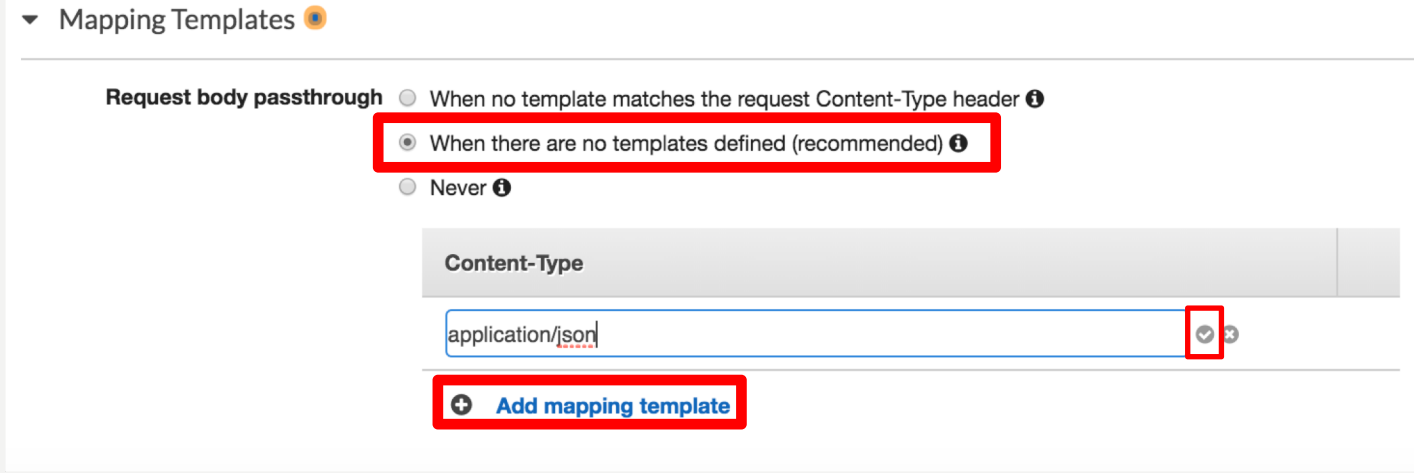
ADD MAPPING TEMPLATE

Set the "Requested body passthrough" to: "When there are no templates defined (recommended)"



Click on "Add mapping template" and enter in: "application/json"

Then click on the check mark next to the input field



ADD MAPPING JSON

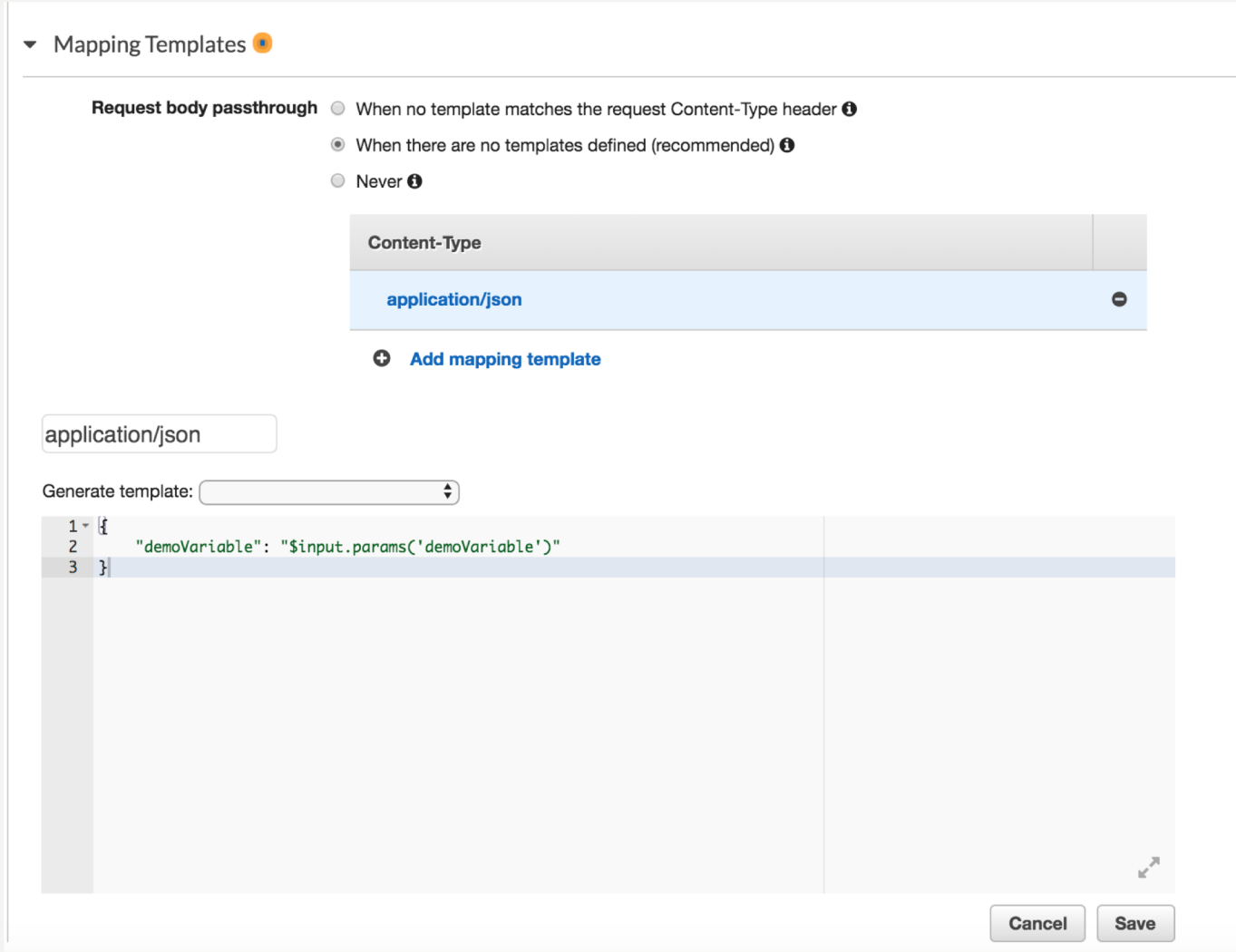
Add a mapping from the variable you wish to use in your lambda function to the same variable name that you will be placing on the URL as a parameter.

Example:

URL: …/demo-stage?demoVariable=test

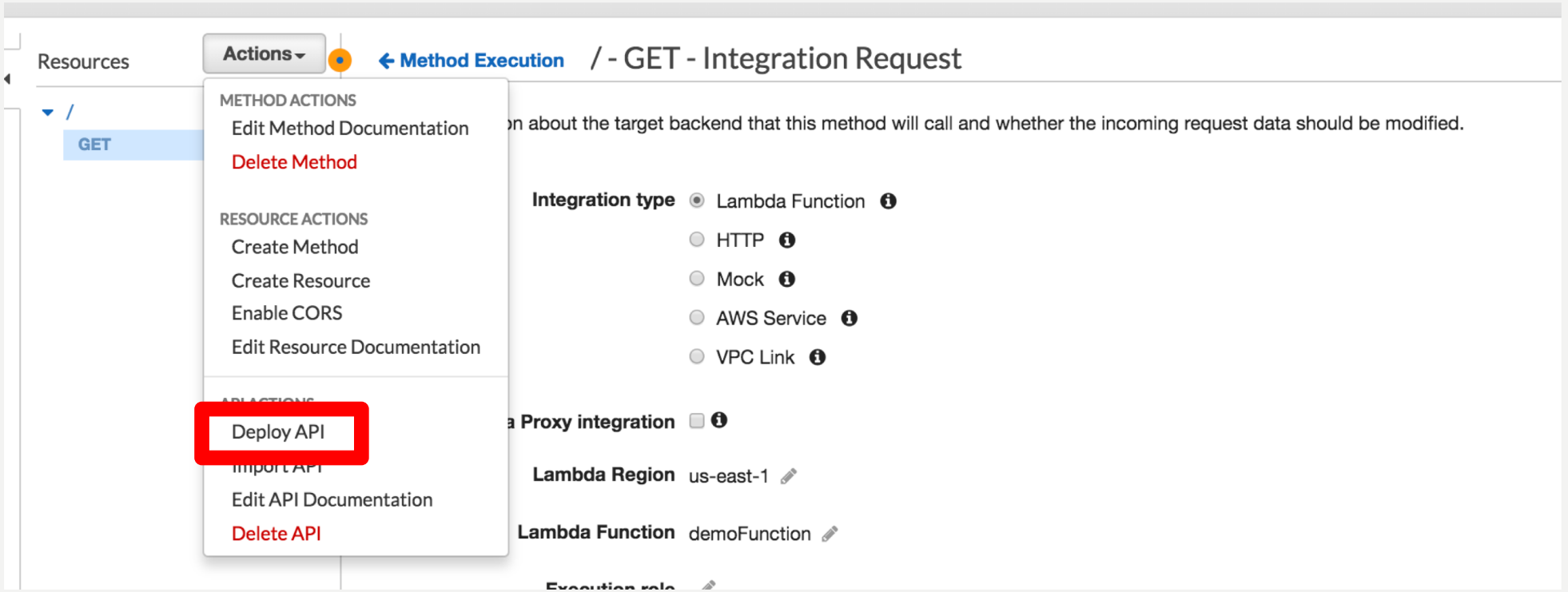
In the above example, demoVariable would be our variable name.

Click "Save" once you have added your mapping.



DEPLOY YOUR API!

Click on Deploy API on the Actions Dropdown



RETRIEVE THE URL PARAMETER VALUE FROM WITHIN YOUR LAMBDA FUNCTION

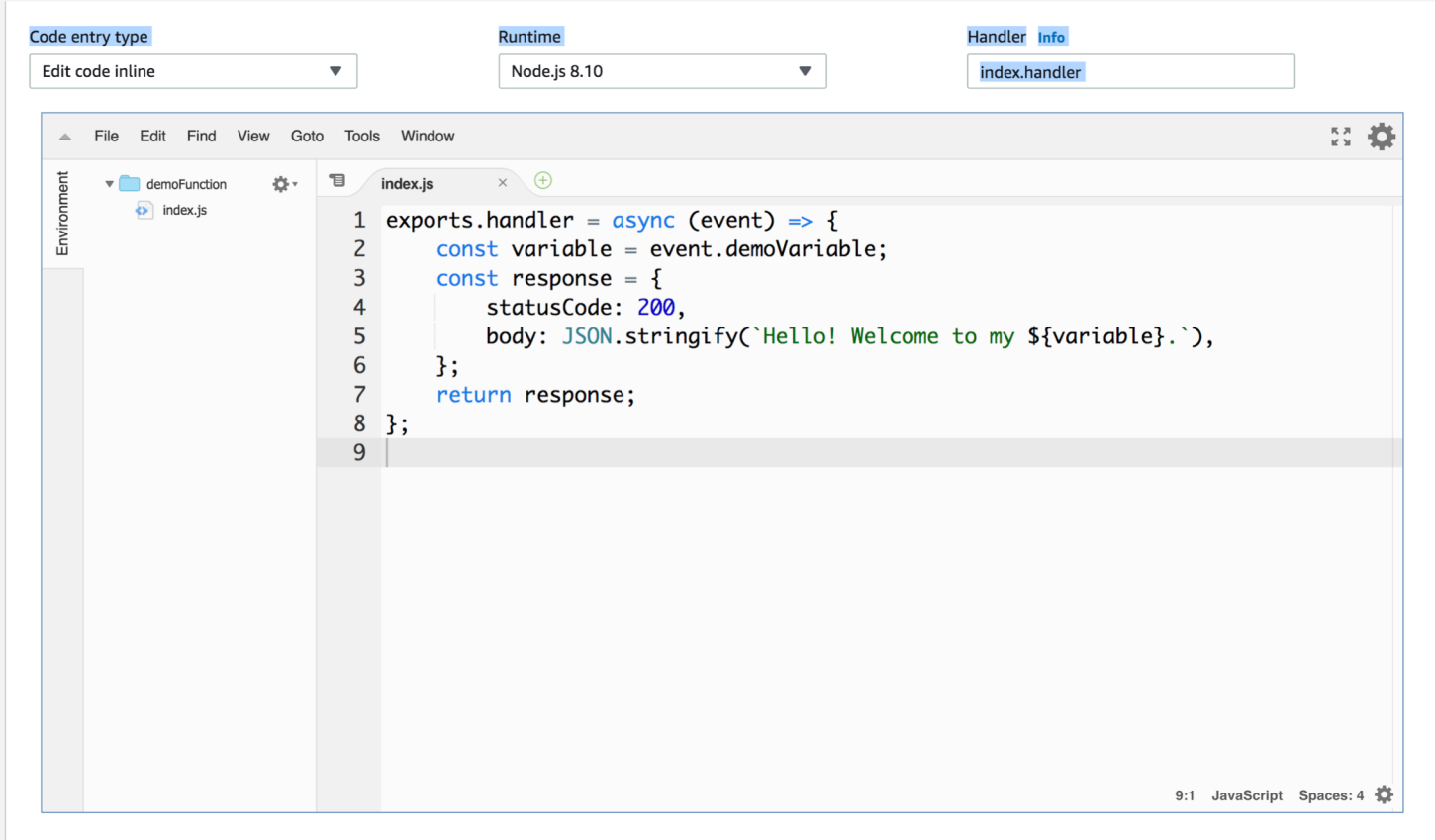
Now that we have the URL parameter based argument set up in our API gateway, let's use that variable in our Lambda Function.

To start, go back to Lambda home: https://console.aws.amazon.com/lambda/home

Then, click on your newly created function

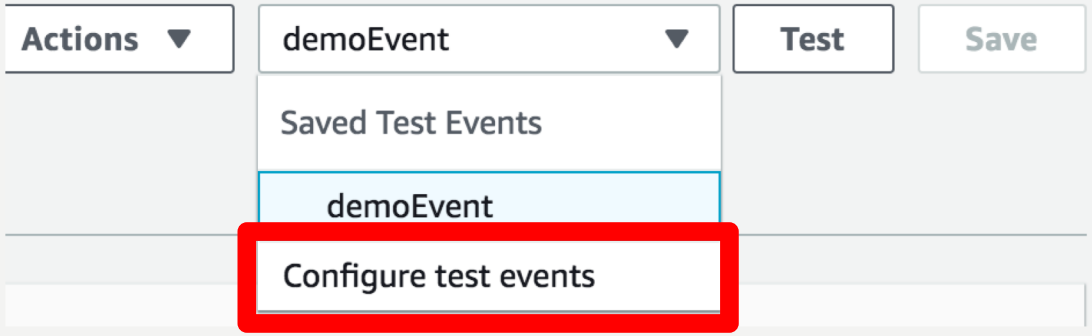
MODIFY THE LOGIC TO RETRIEVE THE VALUE FROM THE EVENT

Pull the value you set up on the API Gateway from the event object



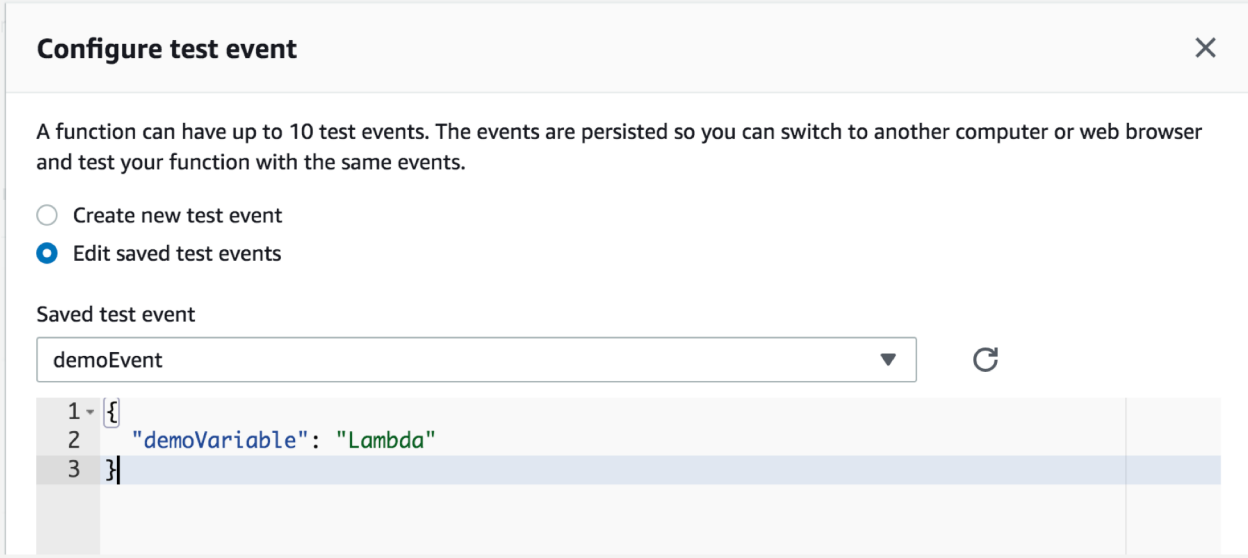
UPDATE YOUR LAMBDA FUNCTION TEST DATA

Click on ”Configure test events”

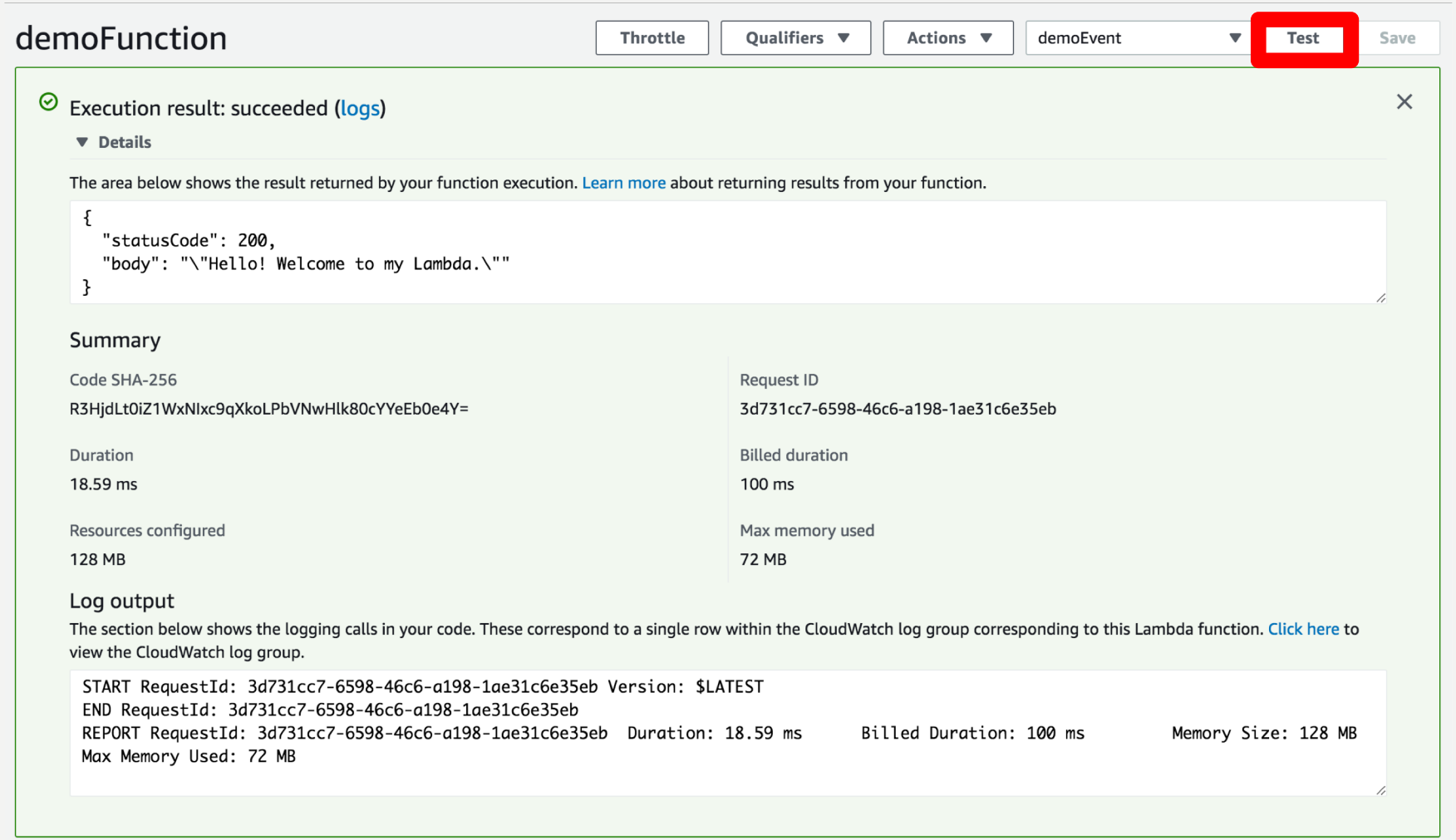


Modify the data and set your variable name to an expected value

Once the value has been set, click save.



TEST YOUR LAMBDA FUNCTION WITH PASSED IN DATA



TEST YOUR UPDATED LAMBDA FUNCTION WITH THE API GATEWAY

When you test your function with the the variable use in the URL Path, you should see that URL variable being used in the body of the response

FOR MORE INFORMATION:

Building Lambda Functions:
https://docs.aws.amazon.com/lambda/latest/dg/lambda-app.html

Creating a REST API in Amazon API Gateway:
https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-create-api.html

Amazon API Gateway API Request and Response Data Mapping Reference:
https://docs.aws.amazon.com/apigateway/latest/developerguide/request-response-data-mappings.html

THANK YOU!